
203.4770: Introduction to Machine Learning

Dr. Rita Osadchy

Outline

1. About the Course
2. What is Machine Learning?
3. Types of problems and situations
4. ML Example

Prerequisites

- The course assumes some basic knowledge of the probability theory, linear algebra, and basic programming skills.
- You should be familiar with:
 - Joint and marginal probability distributions
 - Normal (Gaussian) distribution
 - Expectation and variance
 - Statistical correlation and statistical independence

Probability/
Statistics

- Matrices, vectors, and their multiplication
- Matrix inverse
- Eigen value decomposition

Linear
Algebra

Links to tutorial in the course description in Moodle.

Course Material:

- **Textbooks:**
 - Duda, R. O. Hart, P. E. D., and Stork, G. *Pattern Classification*. New York, NY: Wiley, 2000.
 - T. Hastie, R. Tibshirani, and J. Friedman: "Elements of Statistical Learning", Springer-Verlag, 2001.
 - Pattern Recognition and Machine Learning, by Christopher Bishop. Springer, August 2006.
- Lecture notes and reading material in Moodle.

Final Grade

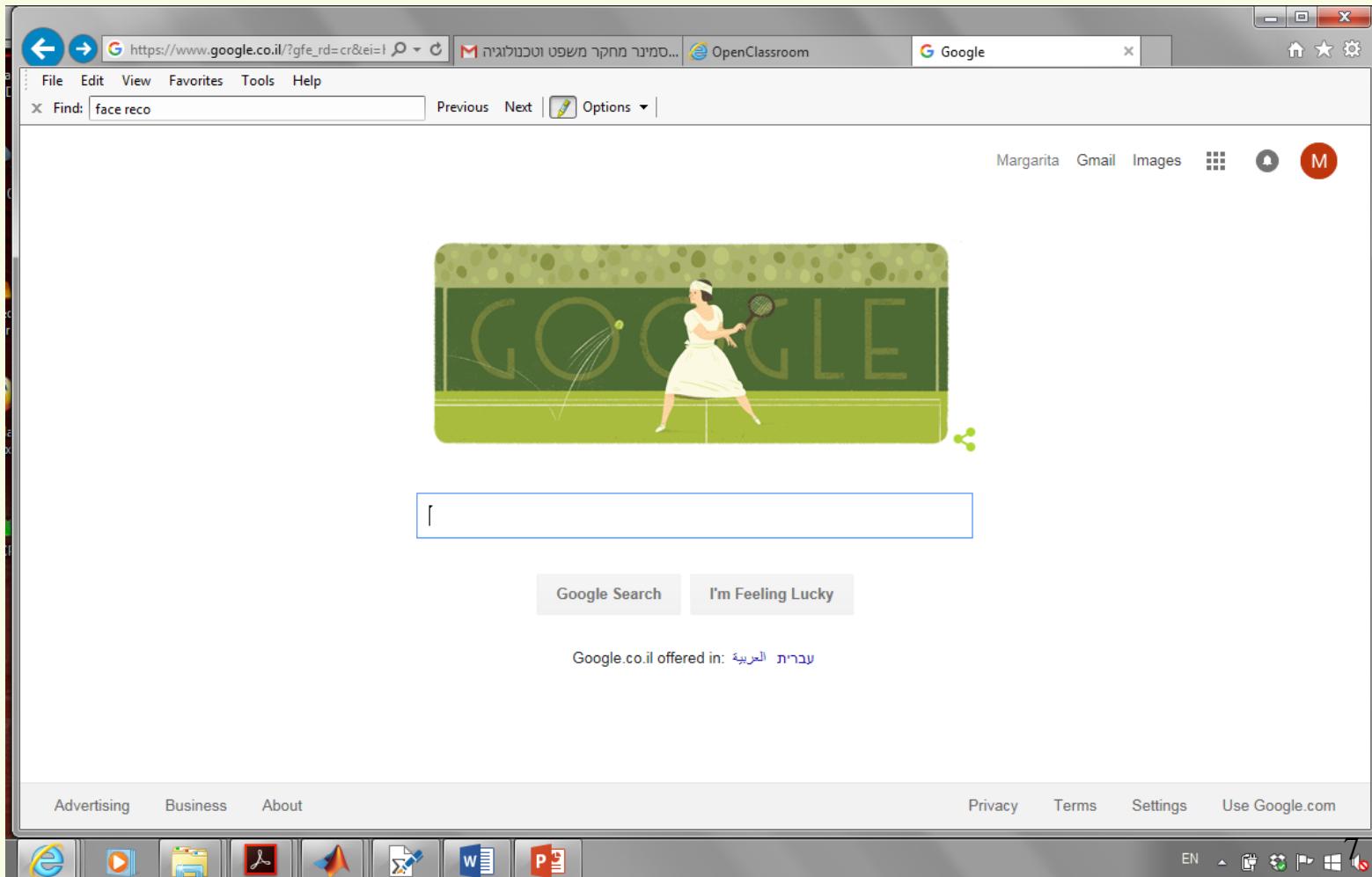
- Home assignments
 - ~20%
- Final Exam
 - ~80%

Home Assignments

- Mostly practical (implement learning algorithms)
- Programming in Matlab
 - Very easy to write code that operates matrices and use plots.
- Submission in pairs (not allowed to change groups after the first assignment)
- **Discussions between groups are allowed, same solutions are not allowed!**

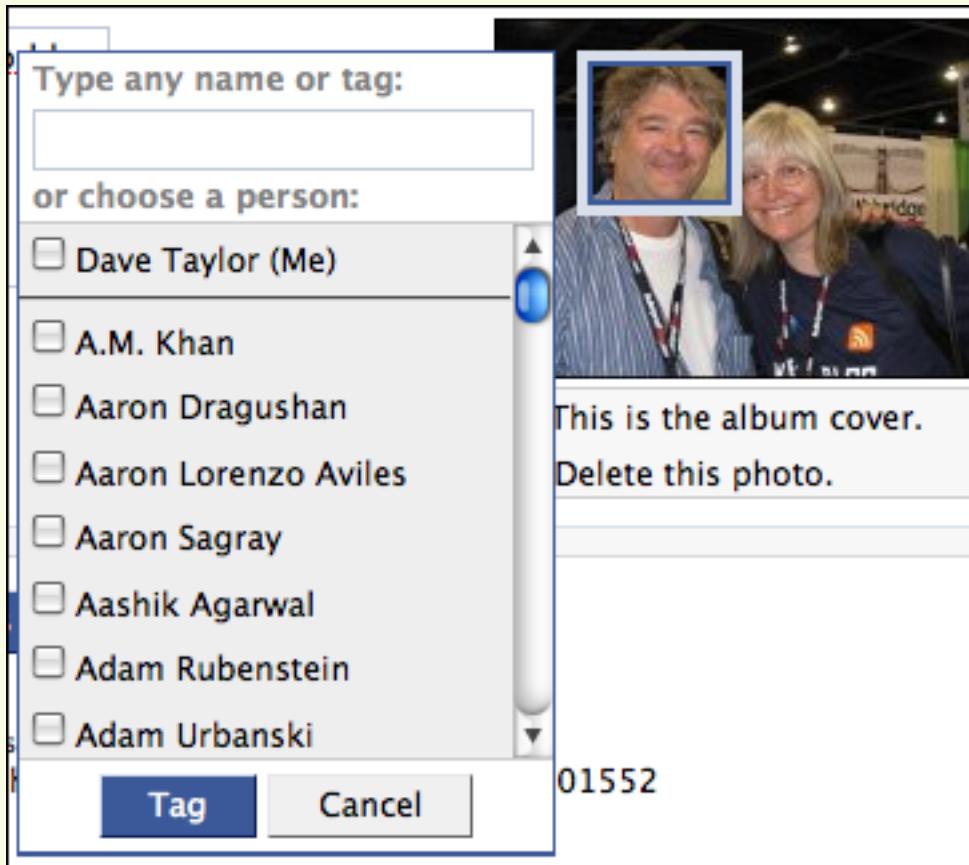
You used ML today...

Google search



You used ML today...

Tagging photos on Facebook or mobile device



You used ML today...

Spam Filters



Why do we need Machine Learning?

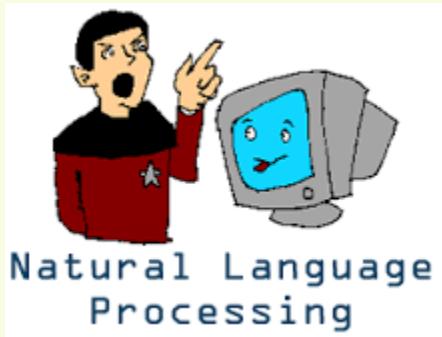
Database mining – large datasets from growth of automation/web:

- Web click data
- Medical records
- Biology
- Engineering
- Finance

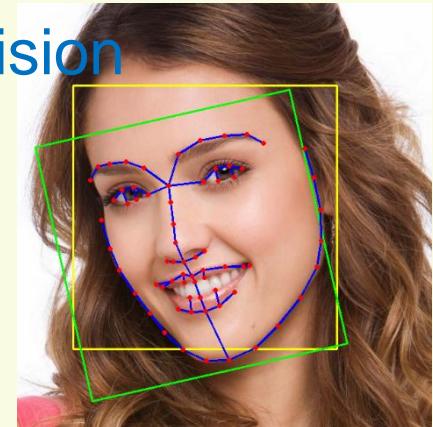
Why do we need Machine Learning?

Applications that cannot be explicitly programmed.

OCR



Computer Vision

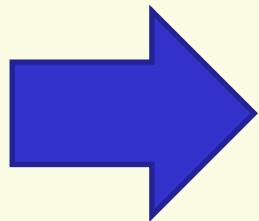
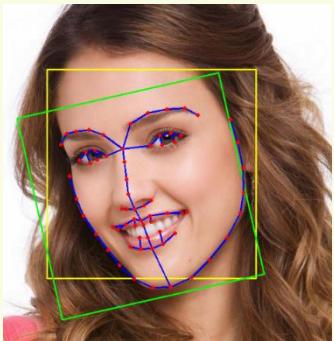


Robot Navigation



Why do we need Machine Learning?

Applications that cannot be explicitly programmed.



Self-driving car



Why do we need Machine Learning?

Self-customizing programs



Product
recommendations



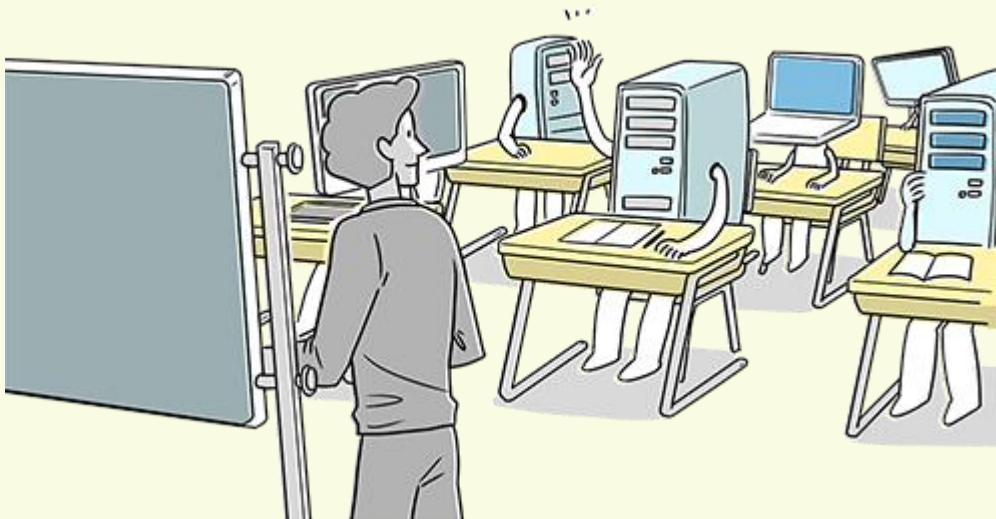
What is Learning?

- Learning is an essential human property
- **Learning**: Acquisition of knowledge, understanding, and ability with experience.
- Learning IS NOT learning by heart



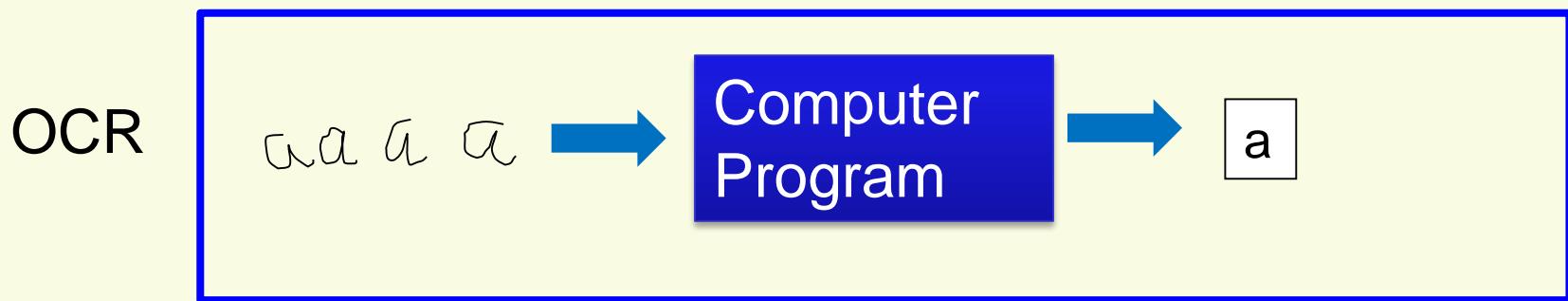
What is Machine Learning?

- Any computer can learn by heart, the difficulty is to make a prediction – **generalize** a behavior to a novel situation.



Machine Learning - Definition

Study of algorithms that improve their performance **P** at some task **T** with experience **E**



T: recognition of a handwritten letter “a” from its image.

E: images of a handwritten “a”.

P: recognition rate.

Types of Machine Learning Problems

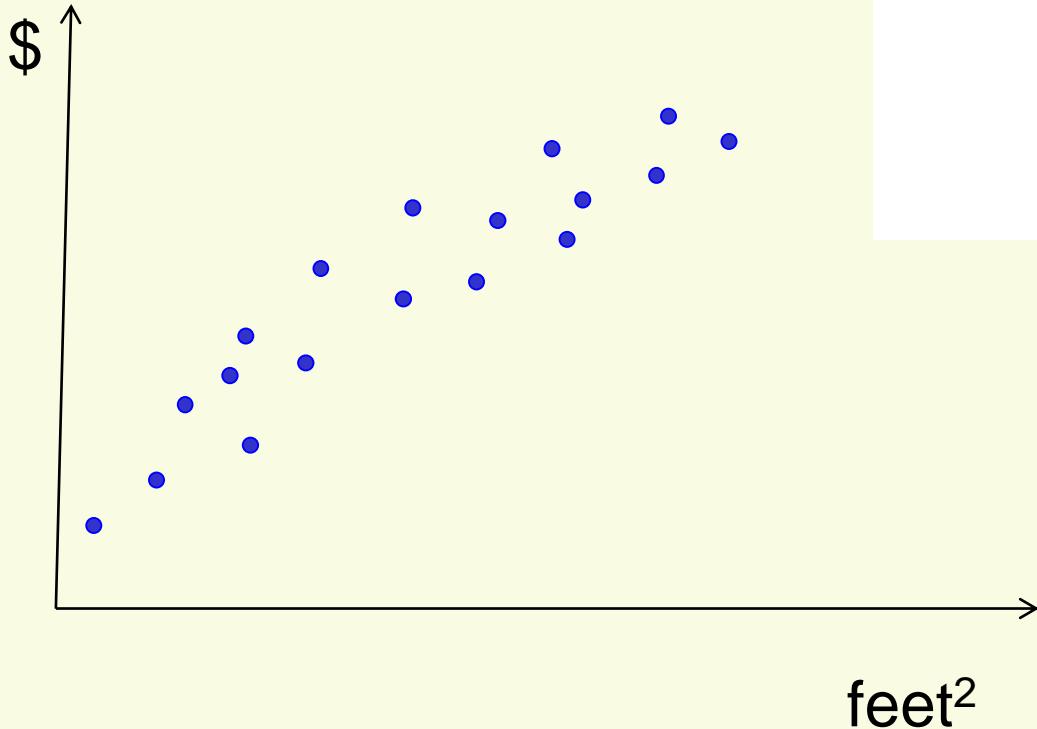
- Supervised learning:
The correct answers are given
- Unsupervised learning:
Find structure in the world
- Other: reinforcement learning, recommender systems...

Supervised Learning

Given a set of training inputs and corresponding outputs (correct answers), produce the “correct” outputs for the new inputs.

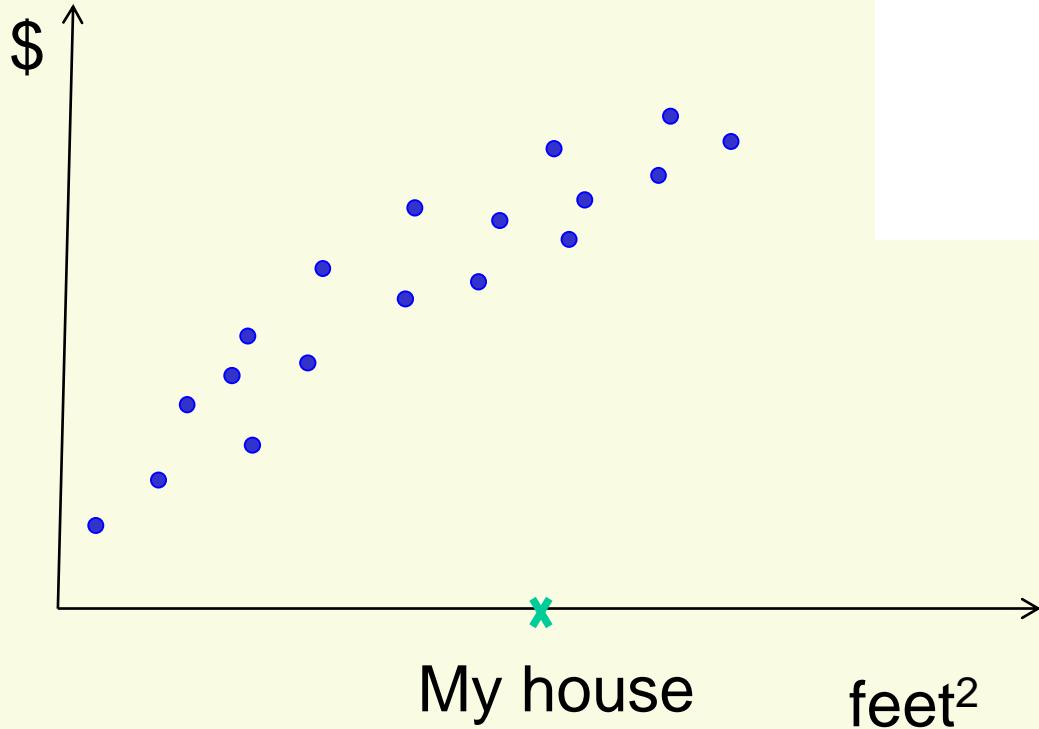
Regression

Housing Prices



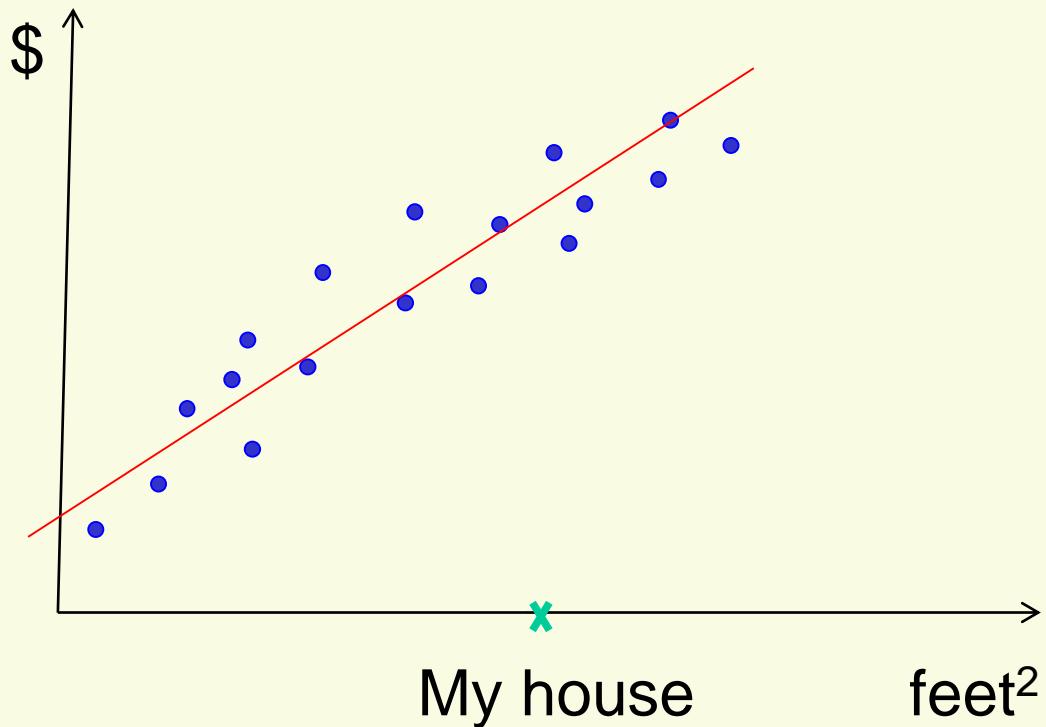
Regression

Housing Prices



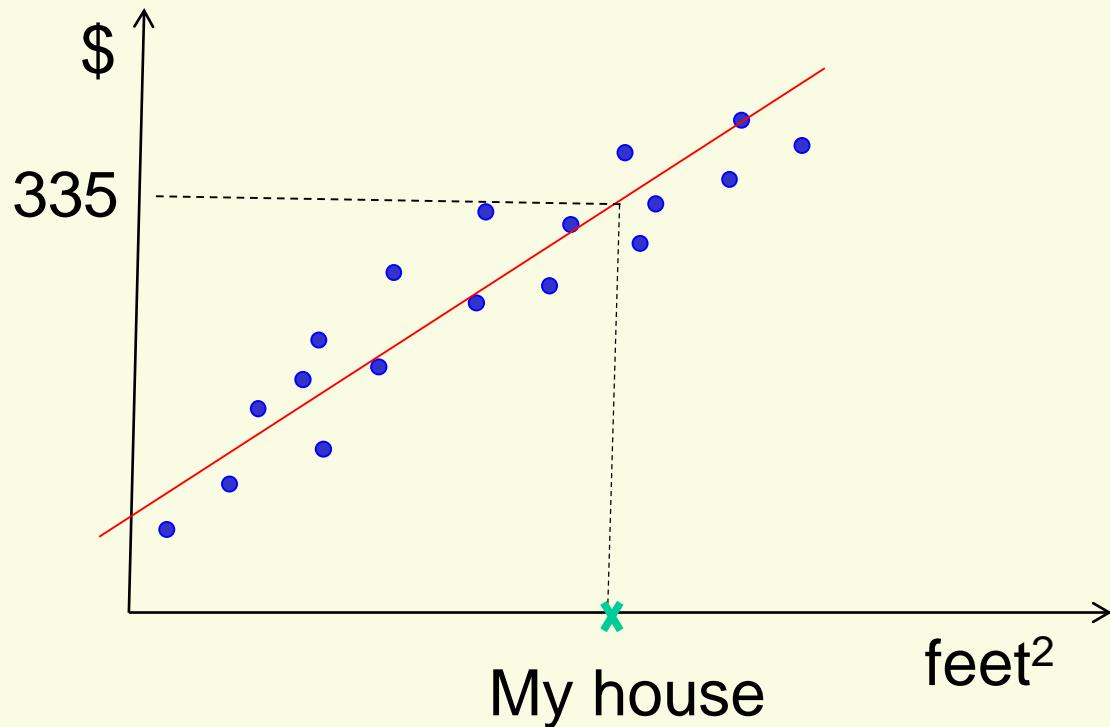
Regression

- Housing Prices



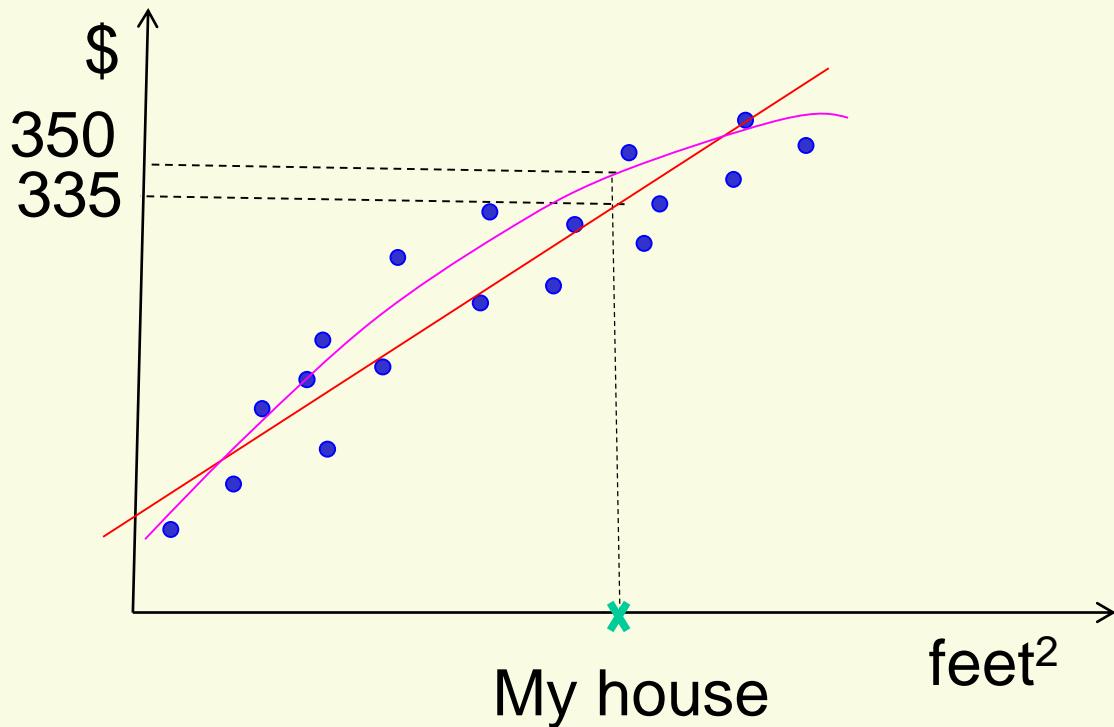
Regression

- Housing Prices



Regression

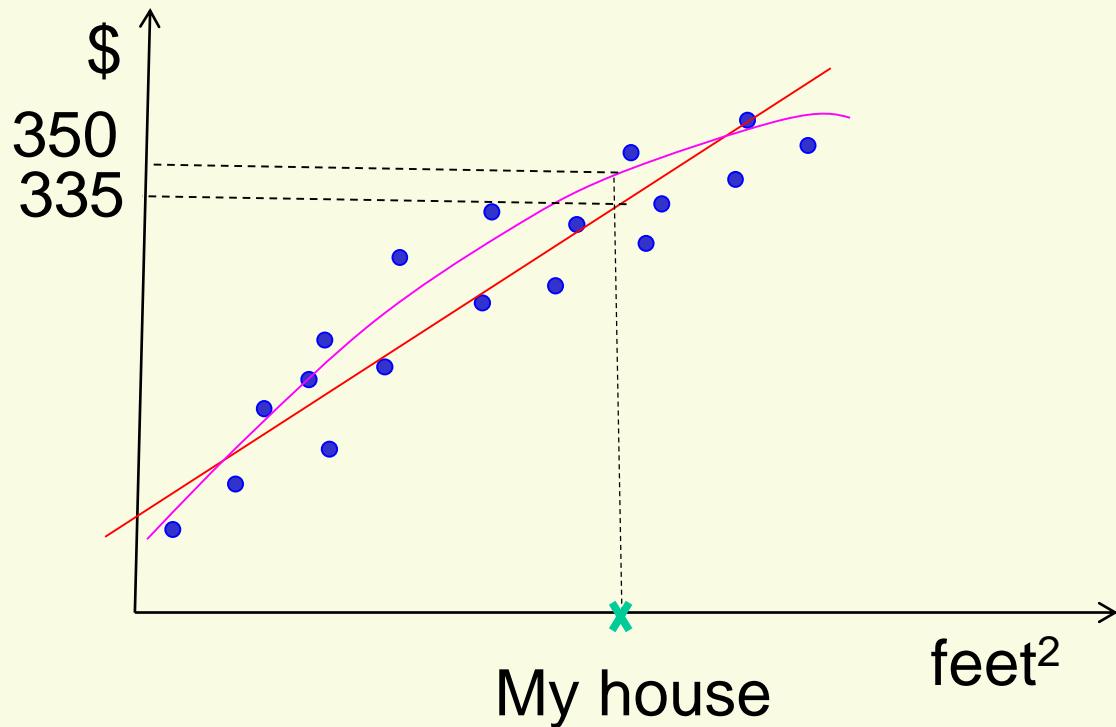
- Housing Prices



Regression

- Housing Prices

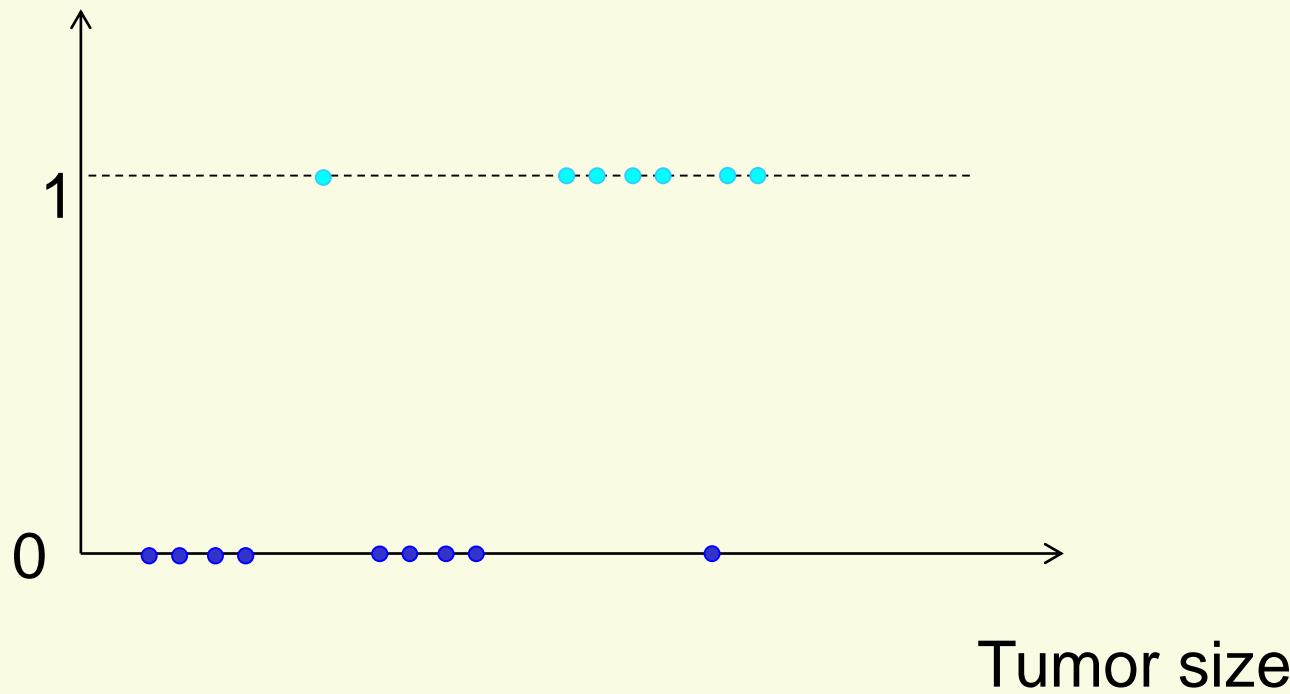
Regression – continues output



Classification

- Tumor Classification

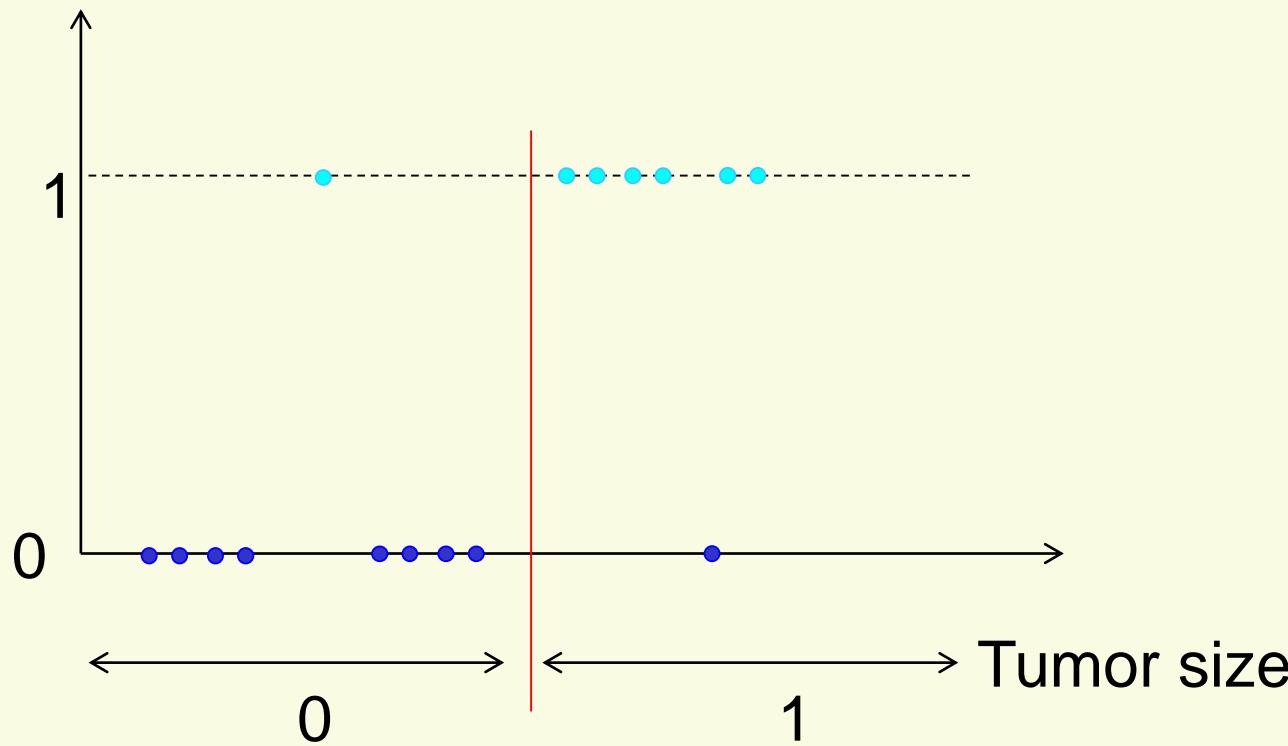
Malignant



Classification

- Tumor Classification

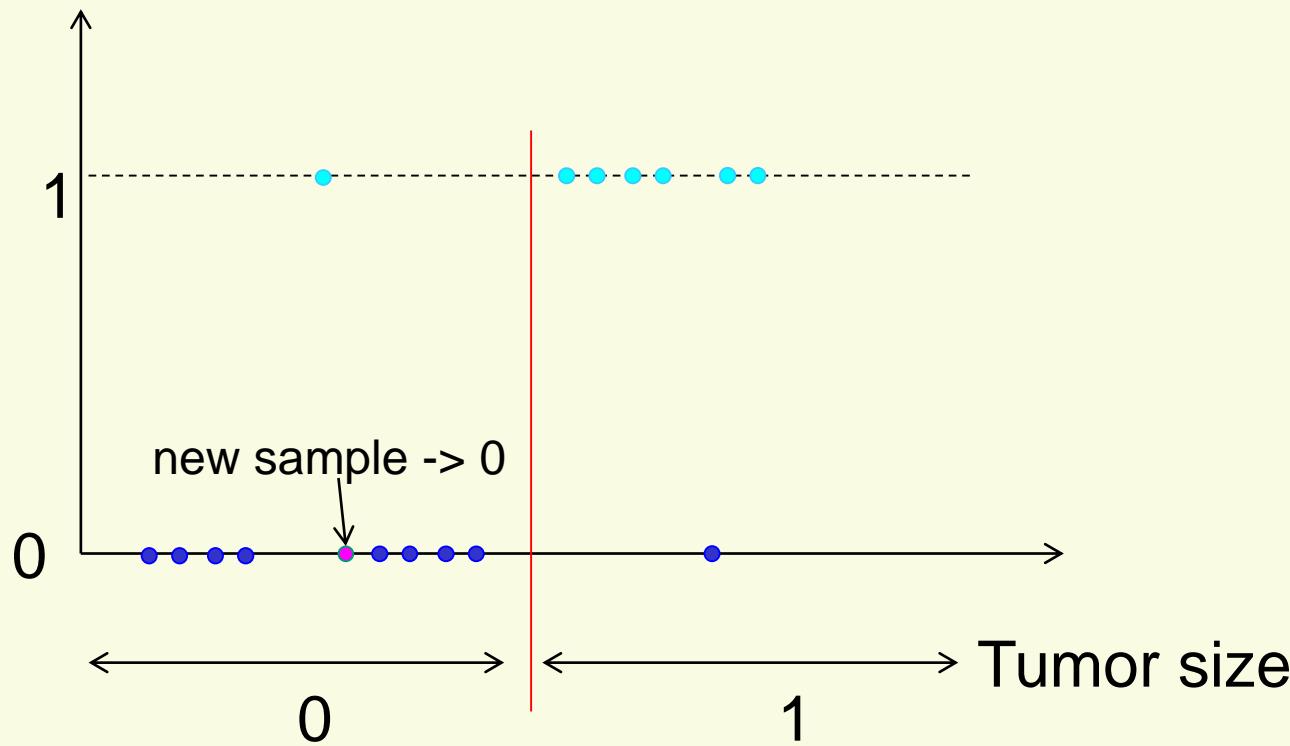
Malignant



Classification

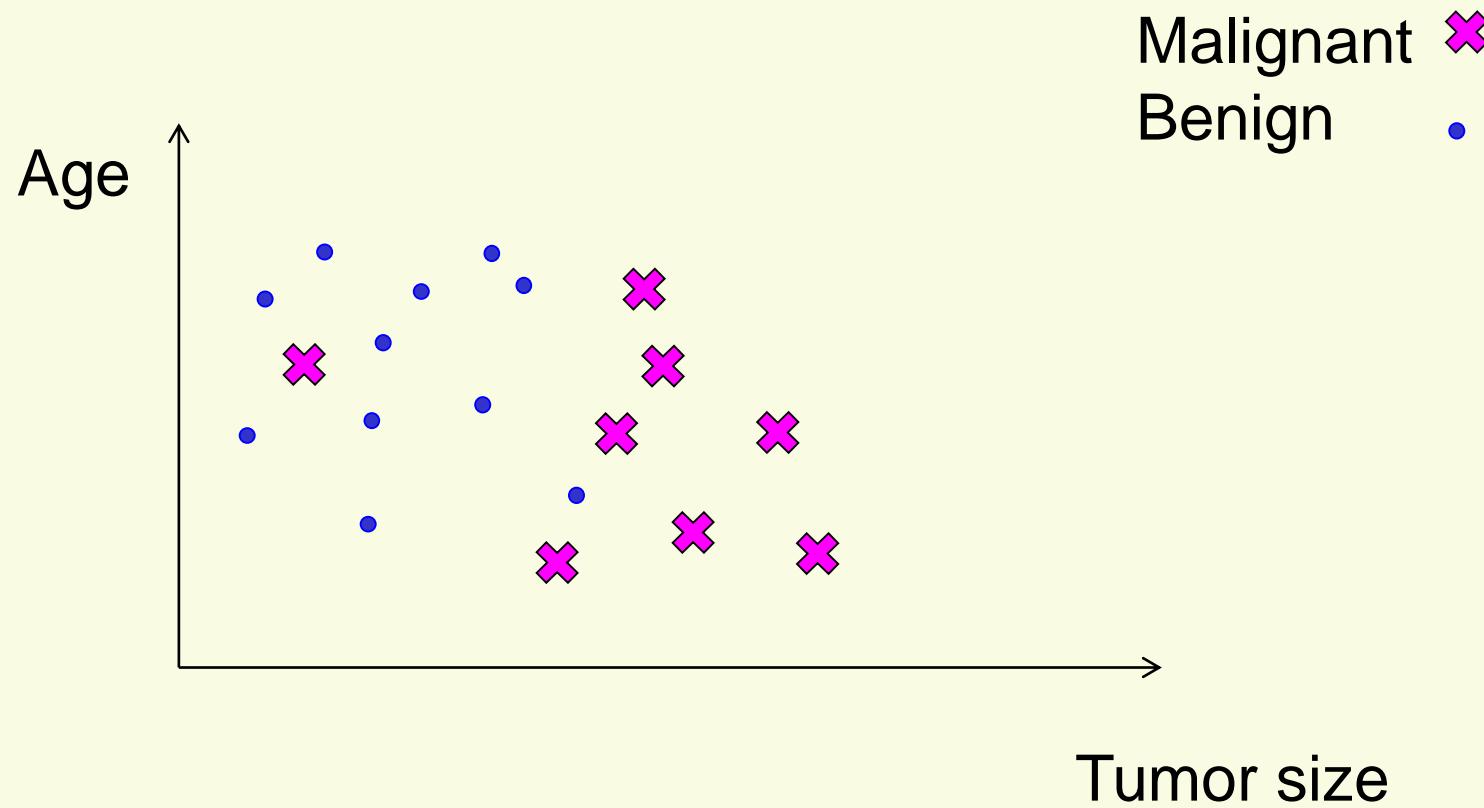
- Tumor Classification

Malignant



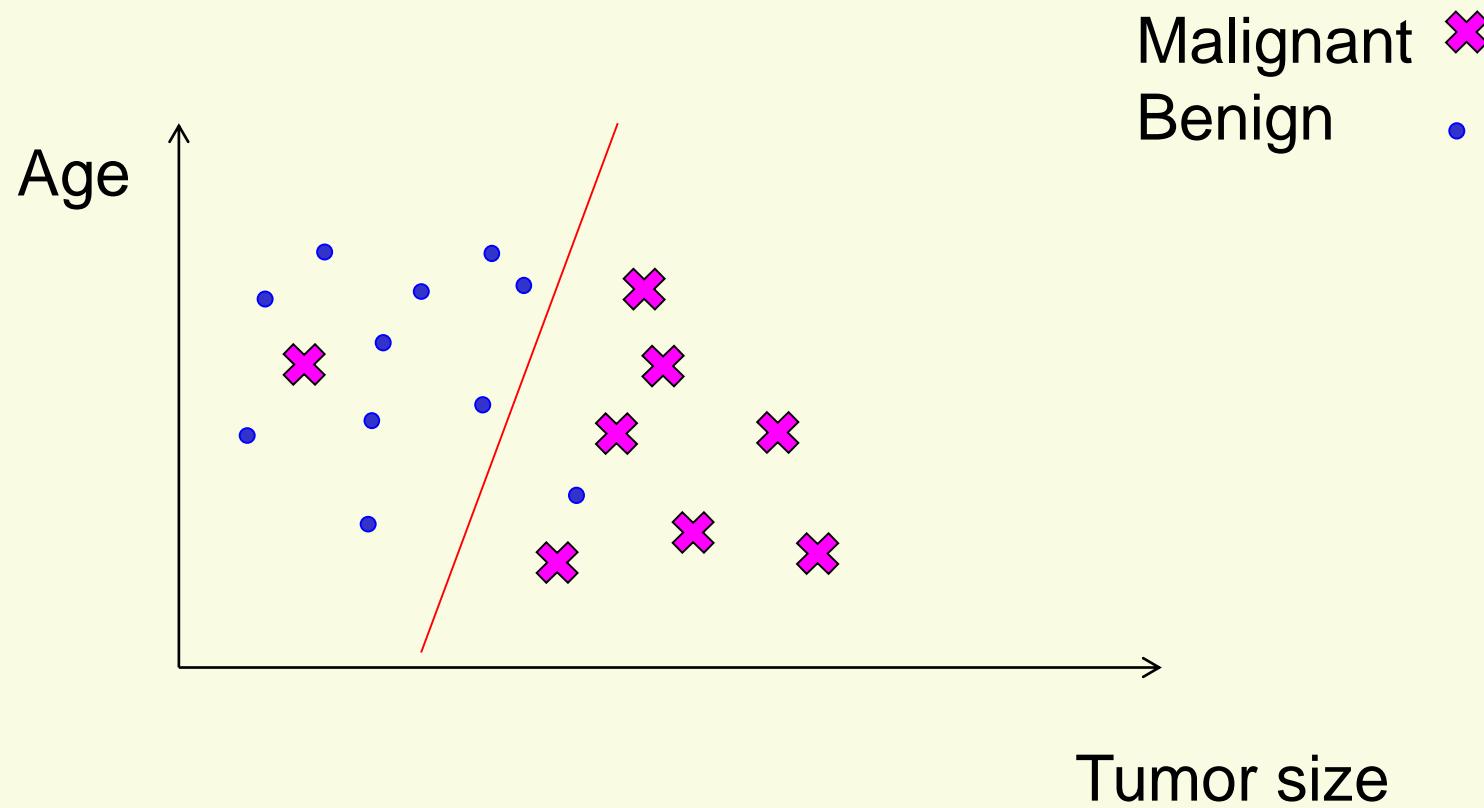
Classification

■ Tumor Classification



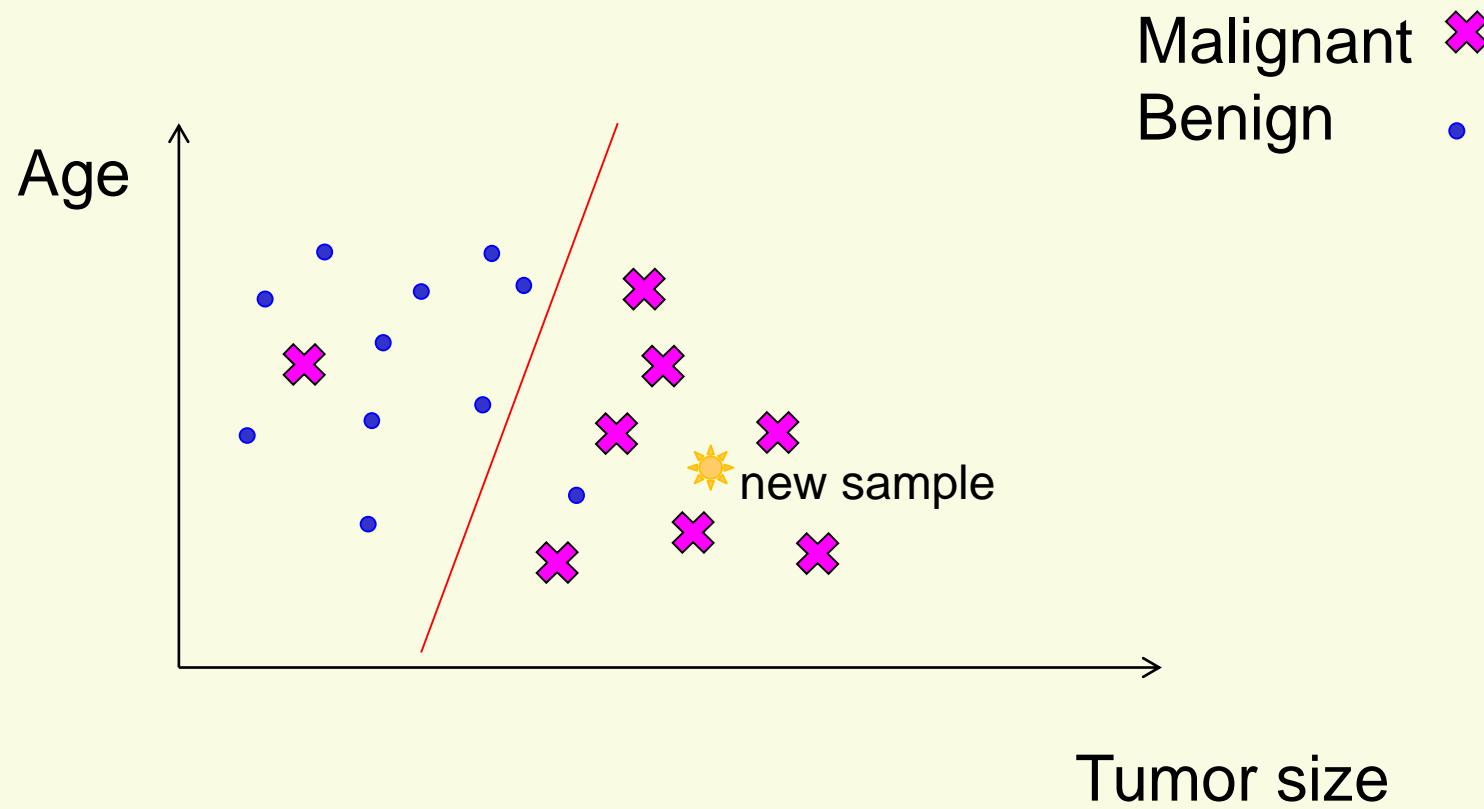
Classification

- Tumor Classification

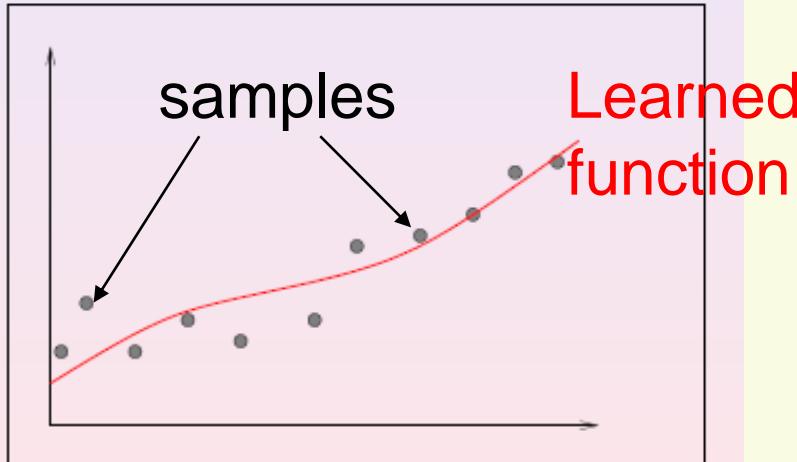


Classification

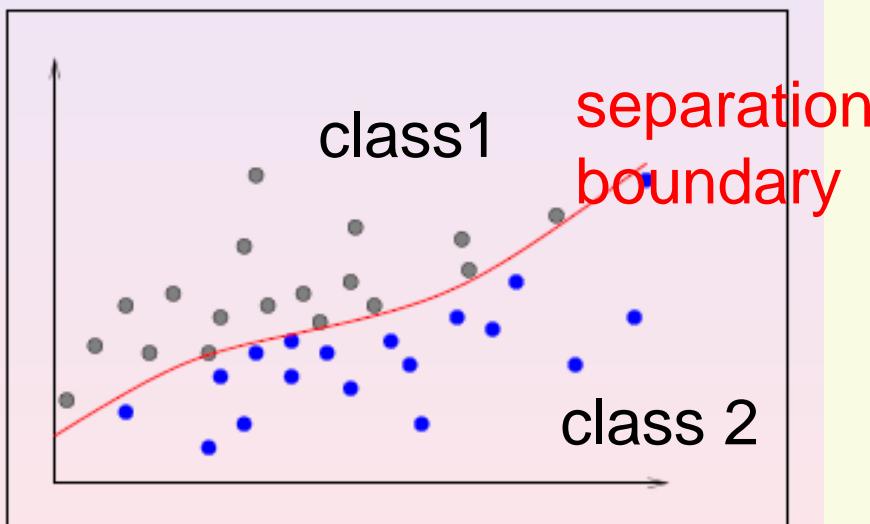
■ Tumor Classification



Two kinds of Supervised Learning

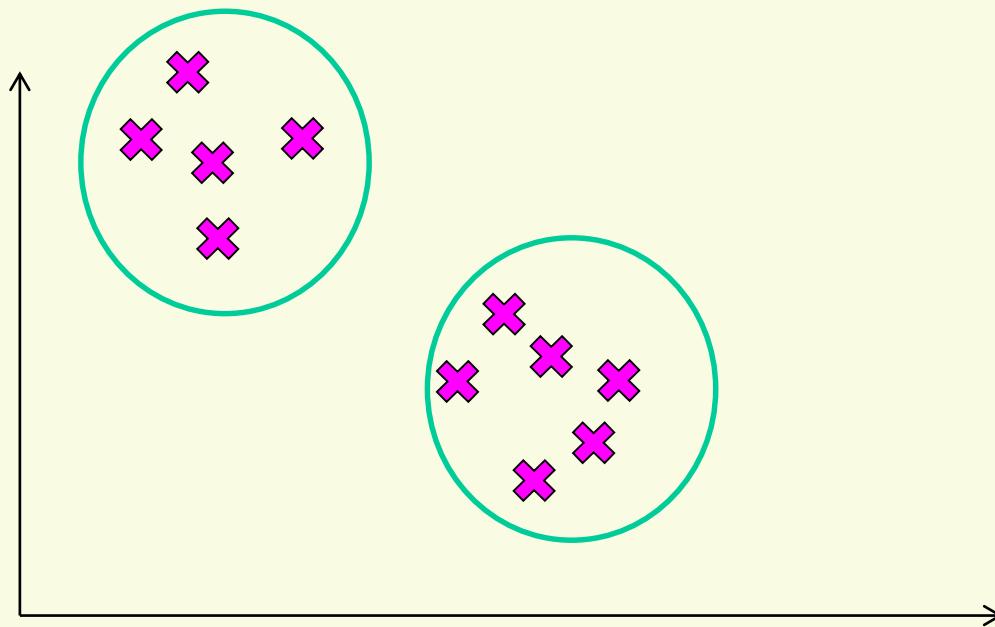


- **Regression:** Learn a continuous input-output mapping from a limited number of examples.



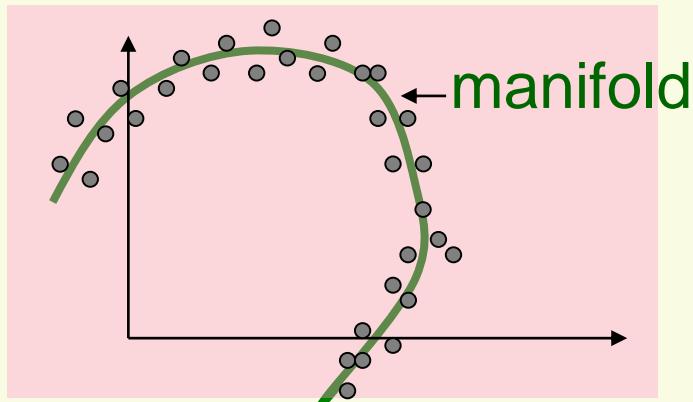
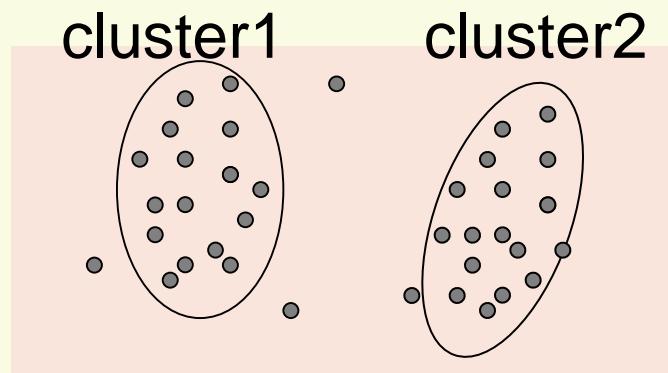
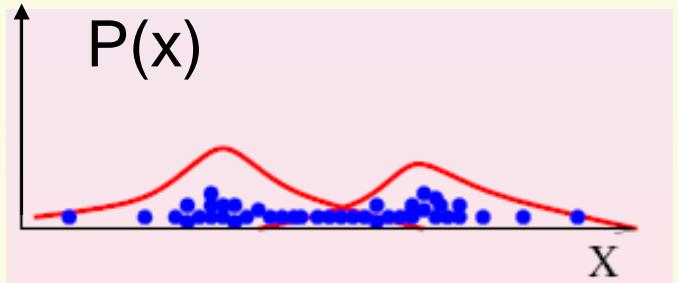
- **Classification:** outputs are discrete variables (category labels). Learn a decision boundary that separates one class from the other.

Unsupervised learning



Given only inputs as training, find structure in the world: discover clusters, manifolds, characterize the areas of the space to which the observed inputs belong.

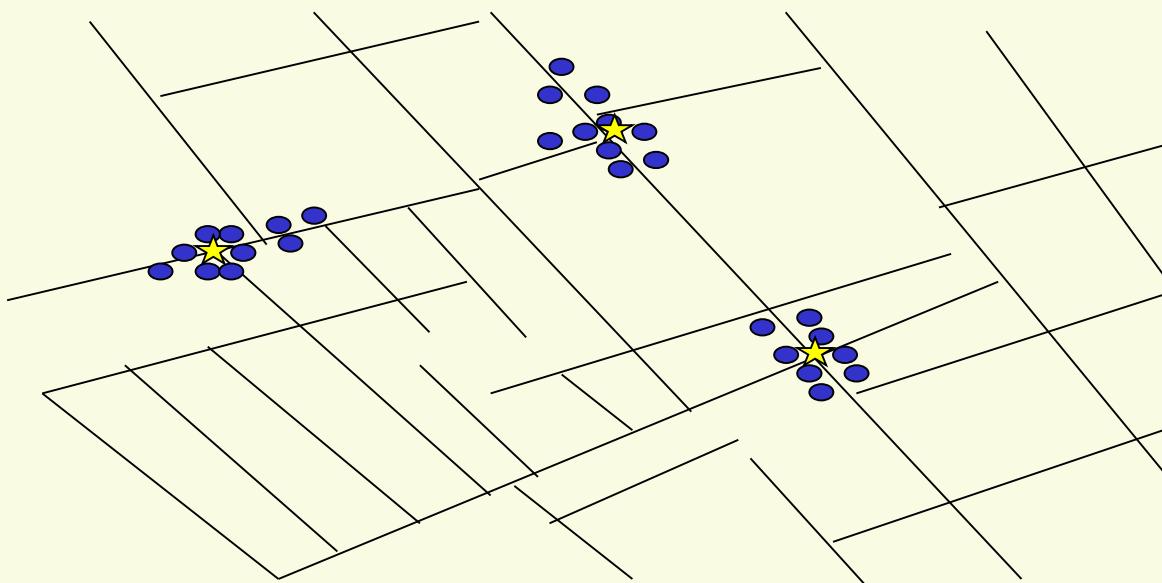
Unsupervised Learning



- **Density Estimation.** Find a function f such $f(X)$ approximates the probability density of X , $p(X)$, as well as possible.
- **Clustering:** discover “clumps” of points
- **Embedding:** discover low-dimensional manifold or surface near which the data lives.

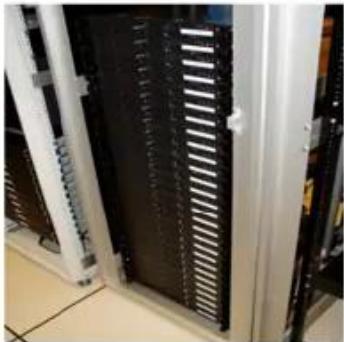
First (?) Application of Clustering

- John Snow, a London physician plotted the location of cholera deaths on a map during an outbreak in the 1850s.
- The locations indicated that cases were clustered around certain intersections where there were polluted wells -- thus exposing both the problem and the solution.

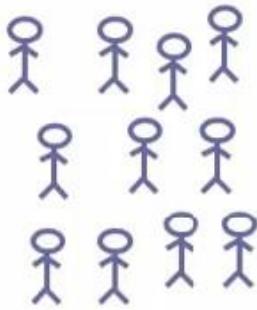


From: Nina Mishra HP Labs

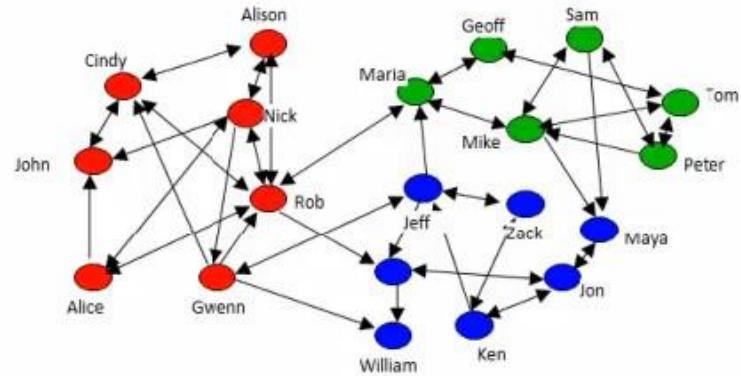
Clustering Applications



Organize computing clusters



Market segmentation.



Social network analysis

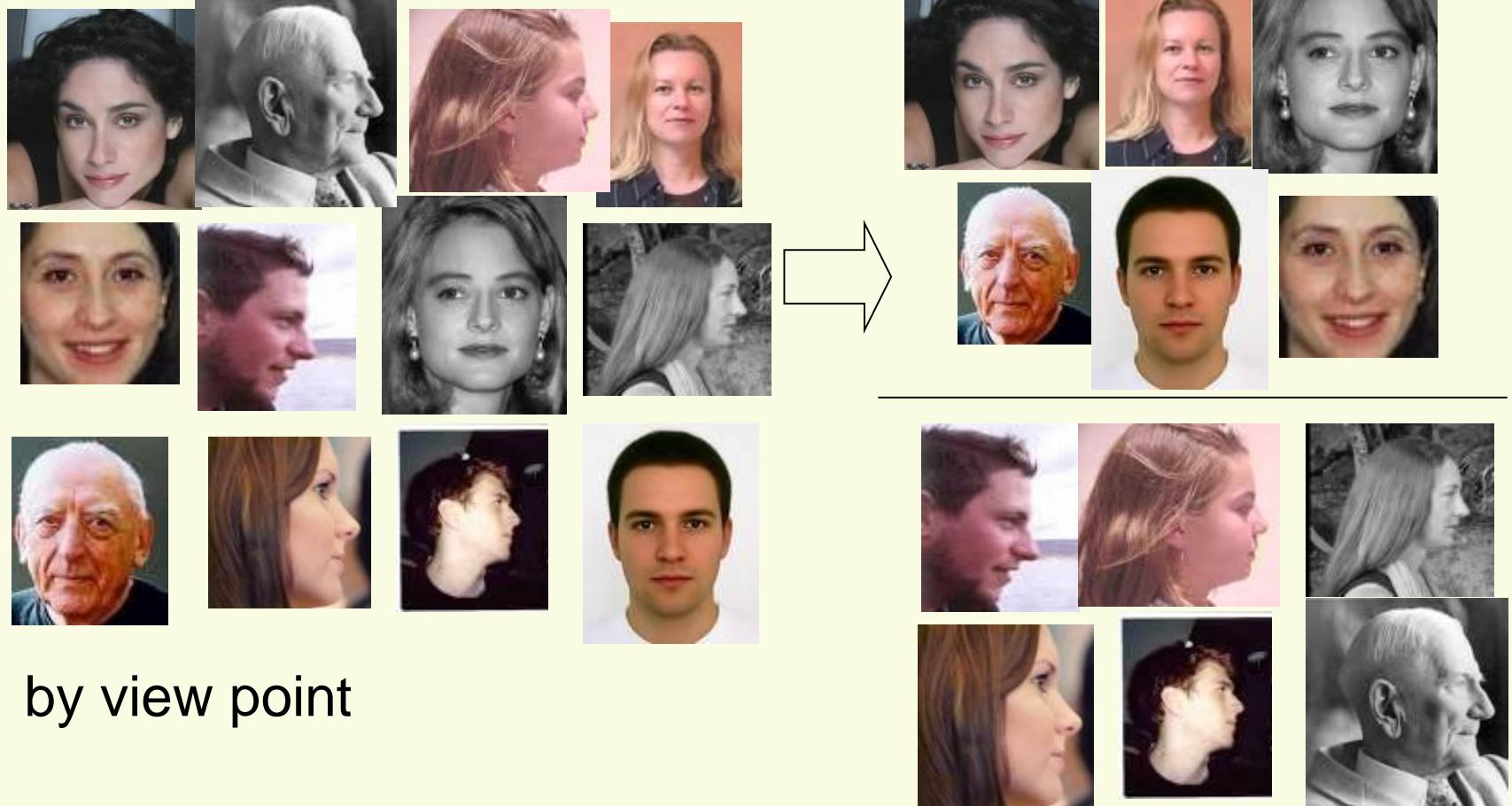


[Image credit: NASA/JPL-Caltech/E. Churchwell (Univ. of Wisconsin, Madison)]

Astronomical data analysis

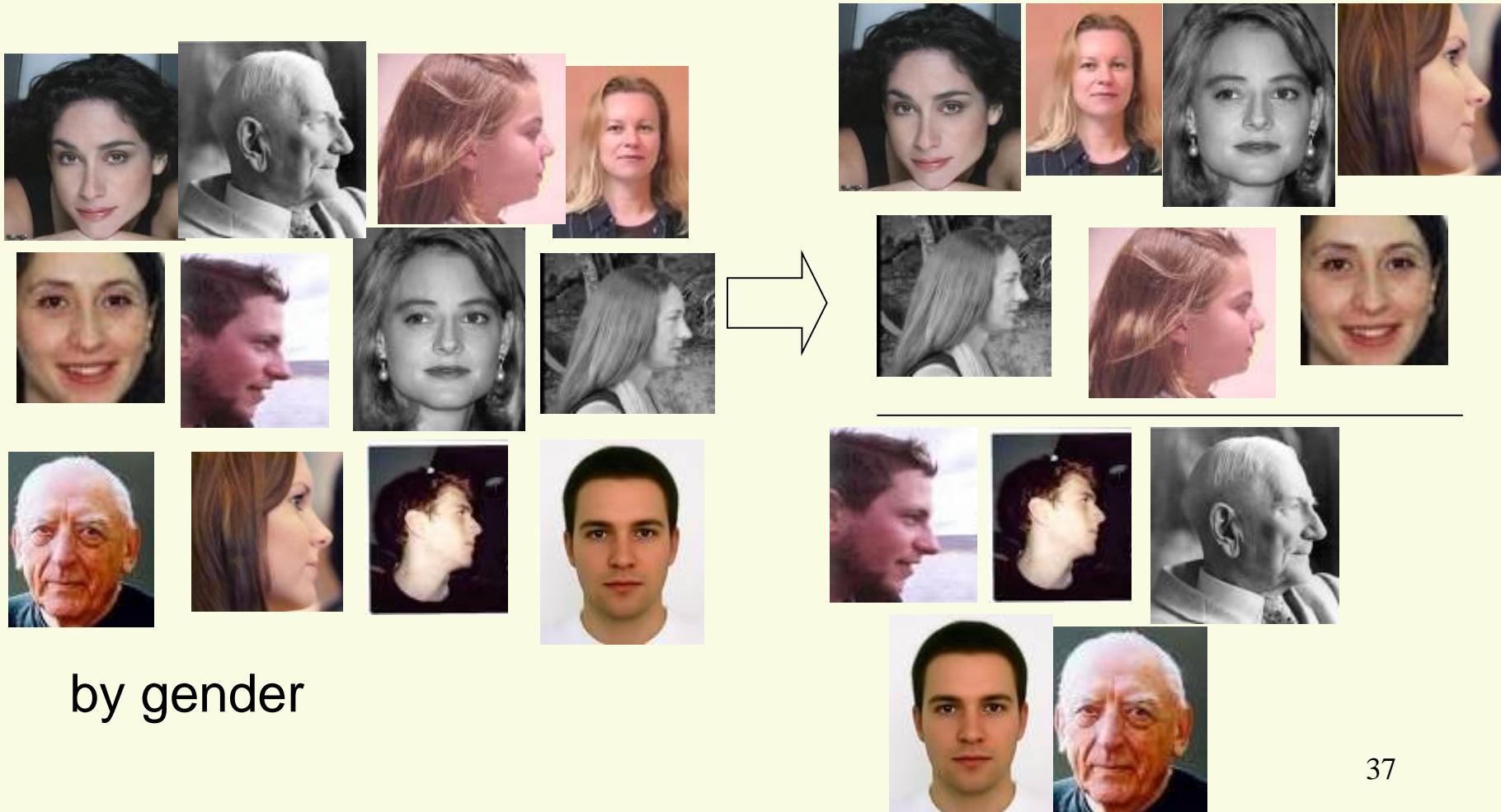
Clustering Example

- Cluster images of faces into two groups



Clustering Example

- Cluster images of faces into two groups

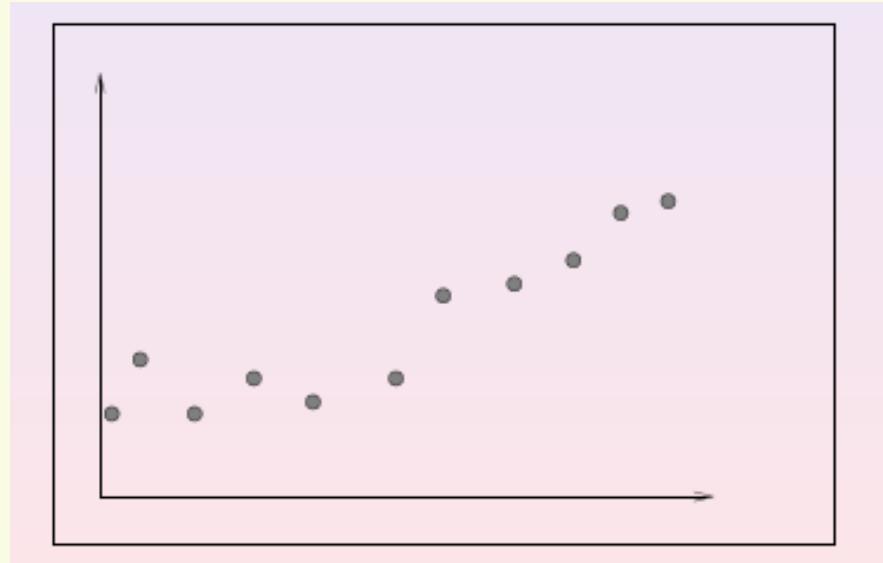


Types of Learning Problems

- Reinforcement learning, where we only get feedback in the form of how well we are doing (For example the outcome of the game).
 - Don't have time to discuss in this course ☹

Why Learning is Difficult?

- Given a finite amount of training data, you have to derive a relation for an infinite domain.
- In fact, there is an infinite number of such relations



Why Learning is Difficult?

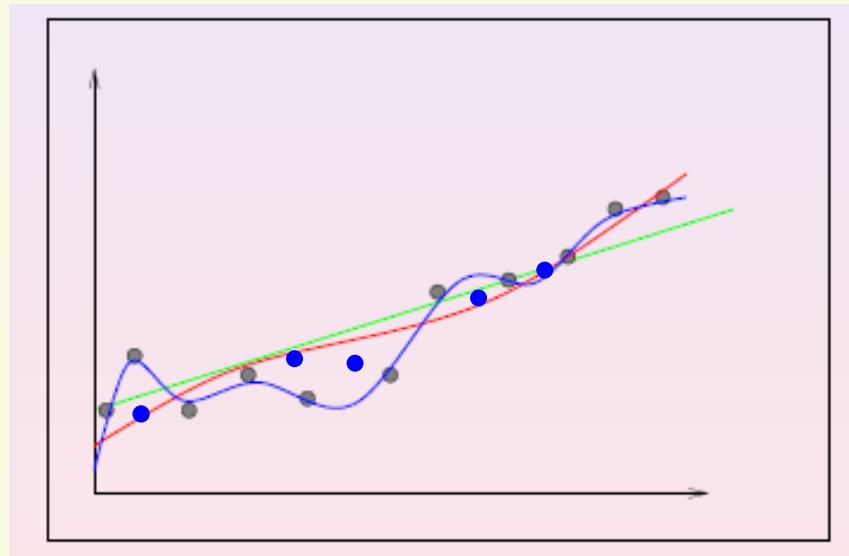
- Given a finite amount of training data, you have to derive a relation for an infinite domain
- In fact, there is an infinite number of such relations



- Which relation is more appropriate?

Why Learning is Difficult?

- Given a finite amount of training data, you have to derive a relation for an infinite domain
- In fact, there is an infinite number of such relations



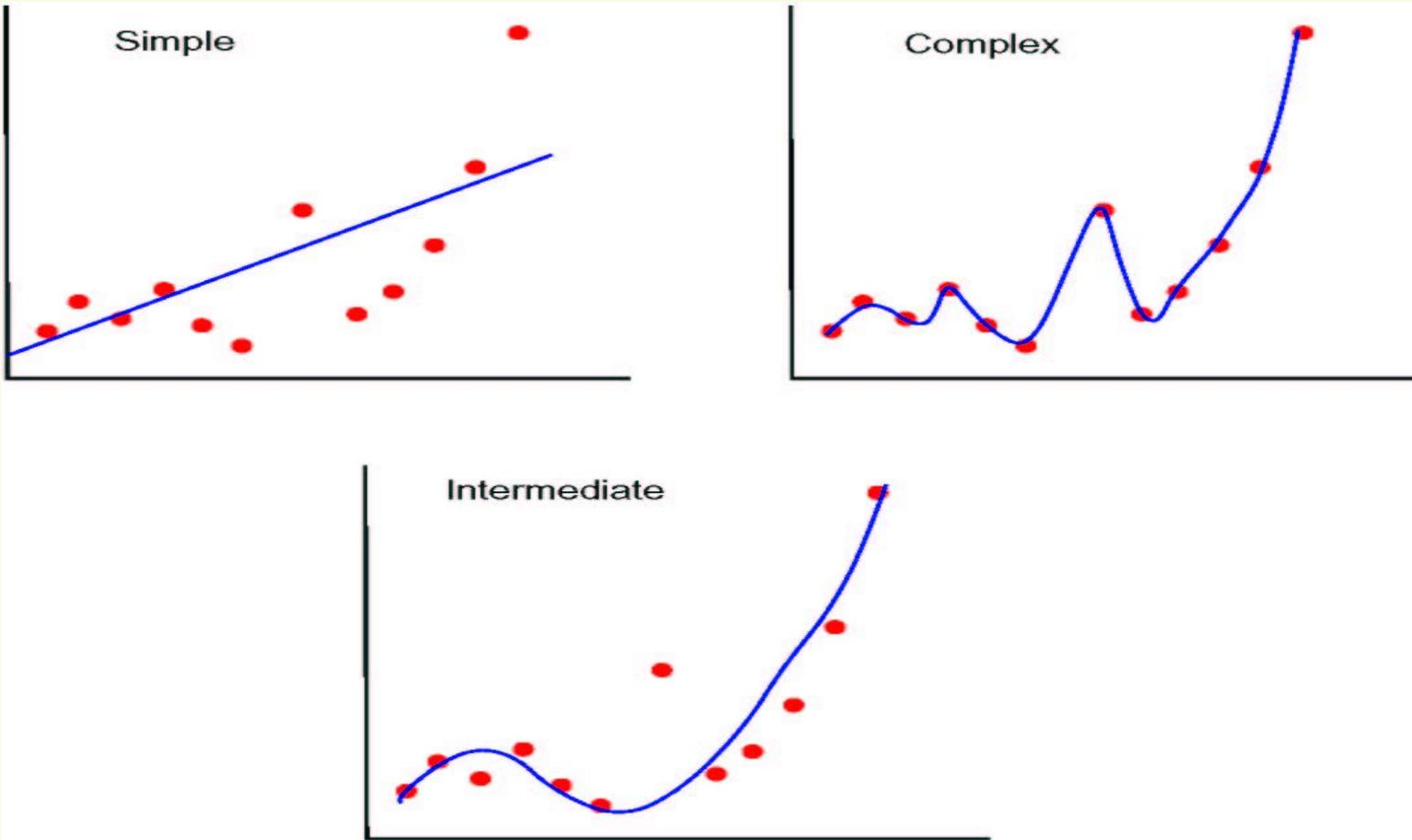
- ... the hidden test points...

Occam's Razor's Principle

- Occam's Razor's Principle(14th century):
One should not increase, beyond what is necessary,
the number of entities required to explain anything
- When **many** solutions are available for a given problem,
we should select the **simplest** one.
- But what do we mean by **simple**?
- We will use prior knowledge of the problem to define what
is a simple solution.

Example of a prior: **smoothness**

Generalization in Regression



Example

- A classification problem: predict the grades for students taking this course.

Example

- A classification problem: predict the grades for students taking this course.
- Key steps:
 - **Data:** what “past experience” can we rely on?

Example

- A classification problem: predict the grades for students taking this course.
- Key steps:
 - **Data:** what “past experience” can we rely on?
 - **Assumptions:** what can we assume about the students or the course?

Example

- A classification problem: predict the grades for students taking this course.
- Key steps:
 - **Data:** what “past experience” can we rely on?
 - **Assumptions:** what can we assume about the students or the course?
 - **Representation:** how do we “summarize” a student?

Example

- A classification problem: predict the grades for students taking this course.
- Key steps:
 - **Data:** what “past experience” can we rely on?
 - **Assumptions:** what can we assume about the students or the course?
 - **Representation:** how do we “summarize” a student?
 - **Estimation:** how do we construct a map from students to grades?

Example

- A classification problem: predict the grades for students taking this course.
- Key steps:
 - **Data:** what “past experience” can we rely on?
 - **Assumptions:** what can we assume about the students or the course?
 - **Representation:** how do we “summarize” a student?
 - **Estimation:** how do we construct a map from students to grades?
 - **Evaluation:** how well are we predicting?

Example

- A classification problem: predict the grades for students taking this course.
- Key steps:
 - **Data:** what “past experience” can we rely on?
 - **Assumptions:** what can we assume about the students or the course?
 - **Representation:** how do we “summarize” a student?
 - **Estimation:** how do we construct a map from students to grades?
 - **Evaluation:** how well are we predicting?
 - **Model selection:** perhaps we can do even better?

Data

- The data we have available (in principle):
 - names and grades of students in past years ML courses
 - academic record of past and current students
- “training” data:

Student	ML	course1	course2	...
Peter	A	B	A	...
David	B	A	A	...

- “test” data:
- Anything else we could use?

Assumptions

- There are many assumptions we can make to facilitate predictions
 1. the course has remained roughly the same over the years
 2. each student performs independently from others

Presentation

- Academic records are rather diverse so we might limit the summaries to a select few courses
- For example, we can summarize the i th student (say Pete) with a vector
$$\mathbf{x}_i = [100 \ 60 \ 80]$$
- The available data in this representation

Training		Test	
Student	ML grade	Student	ML grade
x_1	100	x'_1	?
x_2	80	x'_2	?
...

Estimation

- Given the training data we need to find a mapping from “input vectors” x to “labels” y encoding the grades for the ML course.

Student	ML grade
x1	100
x2	80
...	...

- Possible solution (nearest neighbor classifier):
 - For any student x find the “closest” student x_i in the training set
 - Predict y_i , the grade of the closest student

Evaluation

- How can we tell how good our predictions are?
 - we can wait till the end of this course...
 - we can try to assess the accuracy based on the data we already have (training data)
- Possible solution:
 - divide the training set further into training and validation sets;
 - evaluate the classifier constructed on the basis of only the smaller training set on the new validation set

Model Selection

- We can refine
 - the estimation algorithm (e.g., using a classifier other than the nearest neighbor classifier)
 - the representation (e.g., base the summaries on a different set of courses)
 - the assumptions (e.g., perhaps students work in groups) etc.
- We have to rely on the method of evaluating the accuracy of our predictions to select among the possible refinements

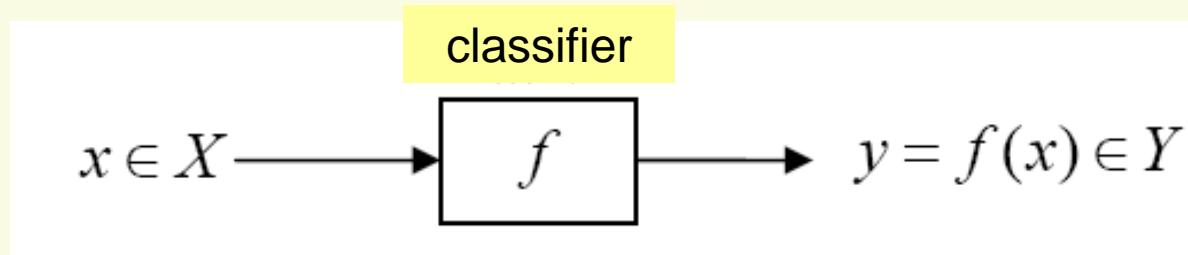
Intro to Classification

Definition of Classification

- A **classifier** is a function or an algorithm that maps every possible input (from a legal set of inputs) to a finite set of decisions.
- X – **input space**, $x \in X$ **sample** from an input space.
- A typical input space is high-dimensional, for example $x = \{x_1, \dots, x_d\} \in R^d$, $d > 1$. We also call x a **feature vector**.
- Ω is a **finite set of categories** to which the input samples belong: $\Omega = \{1, 2, \dots, C\}$.
- $w_i \in \Omega$ are called **labels**.

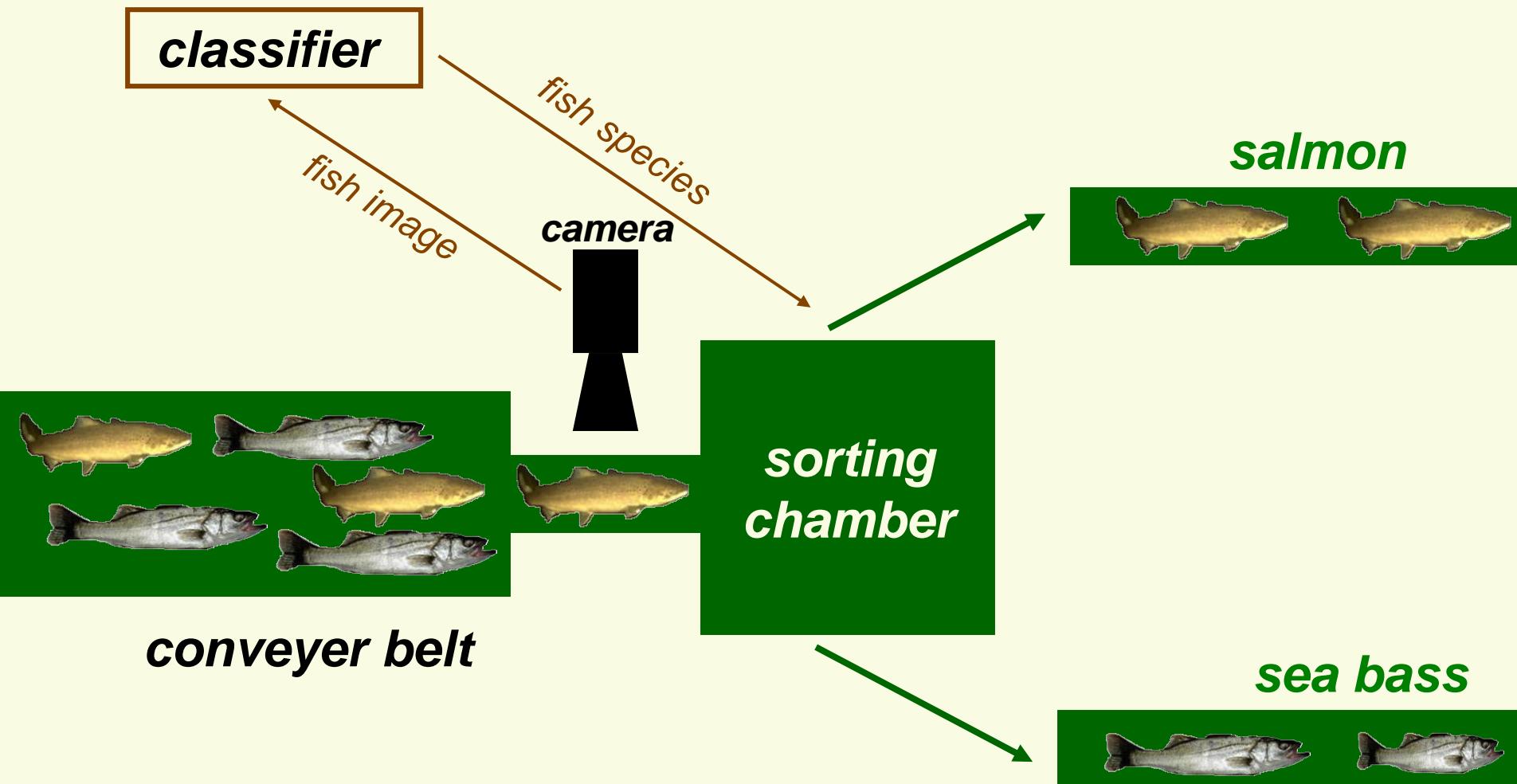
Definition of Classification

- Y is a finite **set of decisions** – the **output set** of the classifier.
- Usually $Y=\Omega$, but it can also contain other decisions, such as “no decision”, “reject” (doesn’t belong to any category from Ω).
- A classifier is a function $f : X \rightarrow Y$



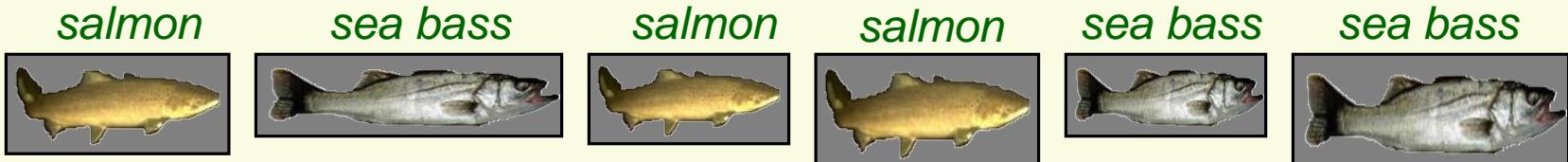
- Classification is also called **Pattern Recognition**.

Our Toy Application: fish sorting



How to design a PR system?

- Collect data and classify by hand



- Preprocess by segmenting fish from background



- Extract possibly discriminating features

- length, lightness, width, number of fins, etc.

- Classifier design

- Choose model

- Train classifier on part of collected data (training data)

- Test classifier on the rest of collected data (test data)

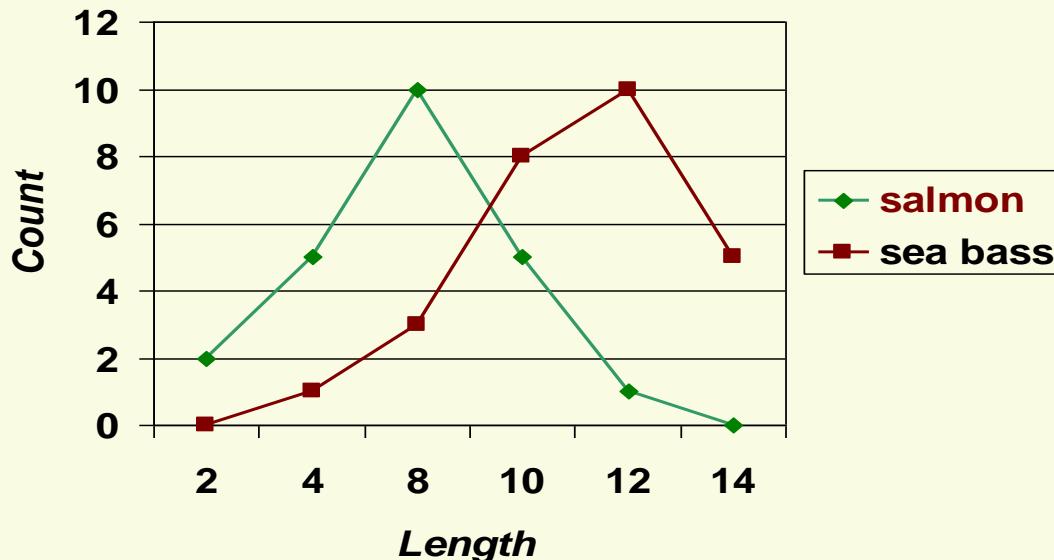
i.e. the data not used for training

- Should classify new data (new fish images) well

Classifier design

- Notice salmon tends to be shorter than sea bass
- Use *fish length* as the discriminating feature
- Count number of bass and salmon of each length

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0



Fish length as discriminating feature

- Find the best length L threshold

fish length < L



classify as salmon

fish length > L



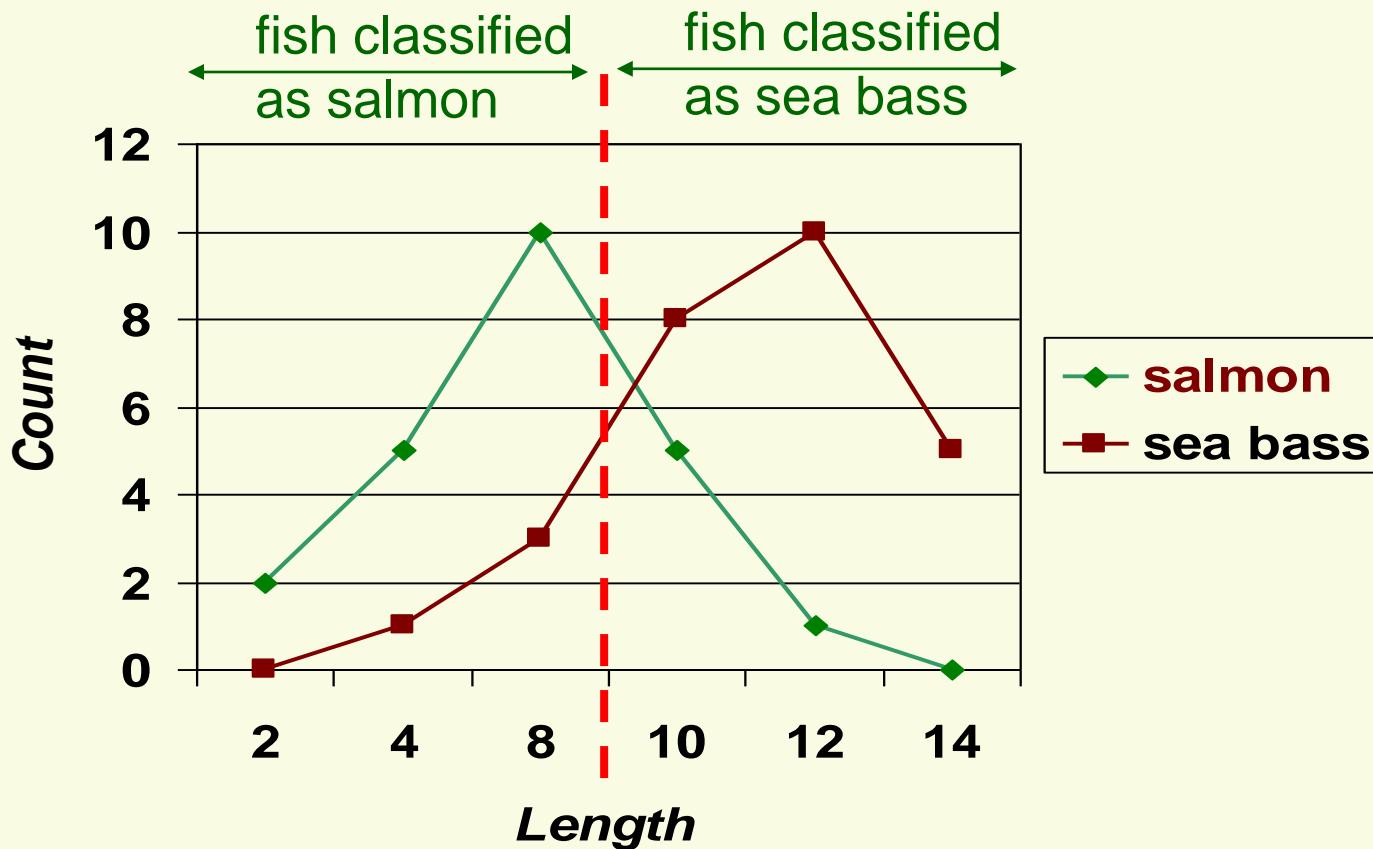
classify as sea bass

- For example, at $L = 5$, misclassified:
 - 1 sea bass
 - 16 salmon

	2	4	8	10	12	14
bass	0	1	3	8	10	5
salmon	2	5	10	5	1	0

- Classification error (total error): $\frac{17}{50} = 34\%$

Fish Length as discriminating feature



- After searching through all possible thresholds L , the best $L=9$, and still 20% of fish is misclassified

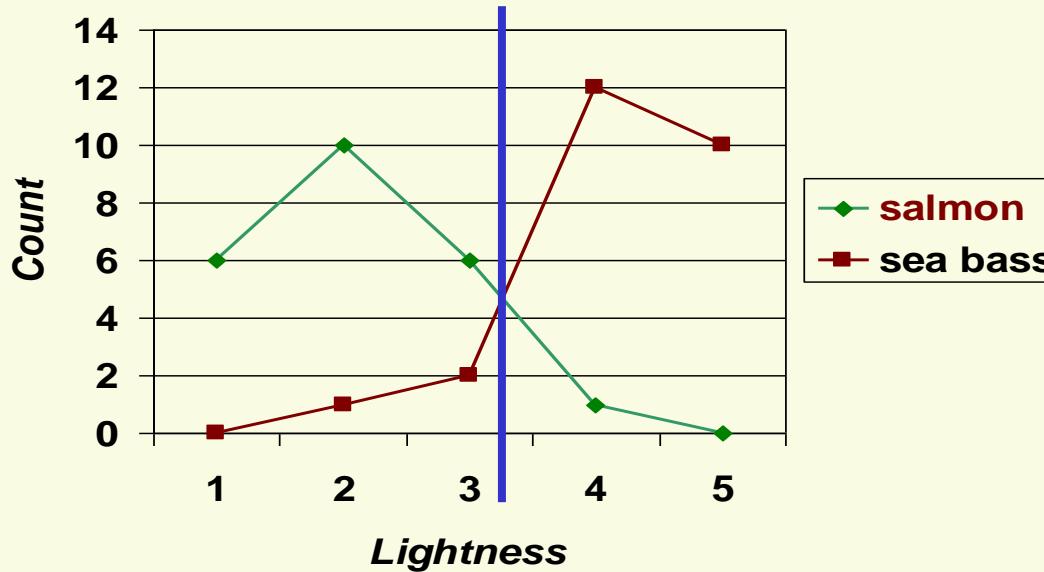
Next Step

- Lesson learned:
 - Length is a poor feature alone!
- What to do?
 - Try another feature
 - Salmon tends to be lighter
 - Try average fish lightness



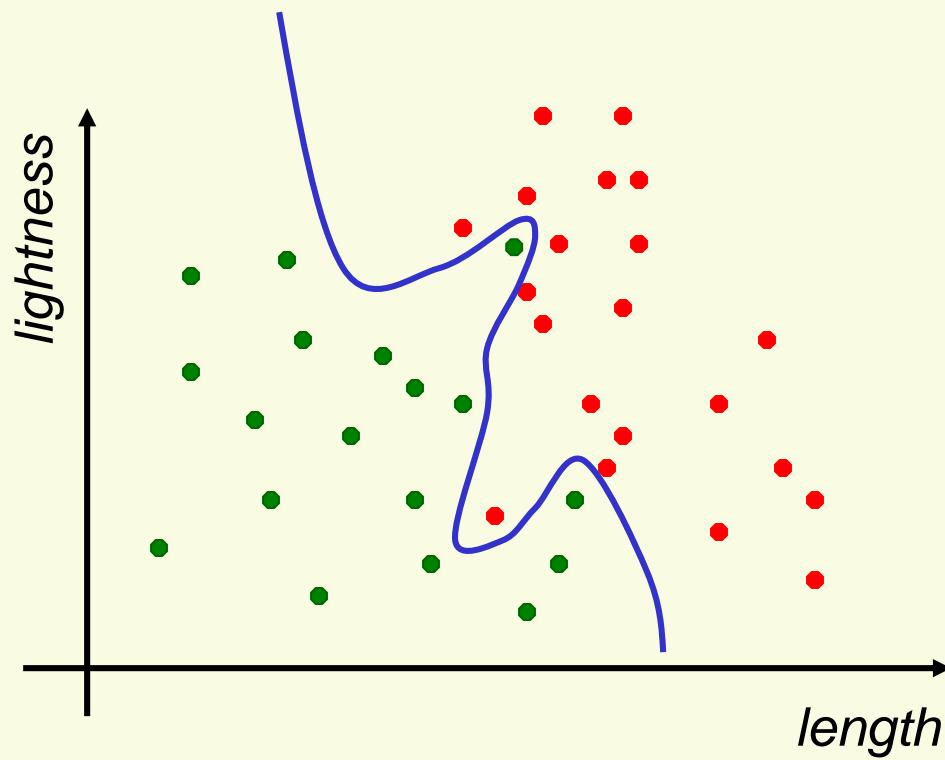
Fish lightness as discriminating feature

	1	2	3	4	5
bass	0	1	2	10	12
salmon	6	10	6	1	0



- Now fish are well separated at lightness threshold of 3.5 with classification error of 8%

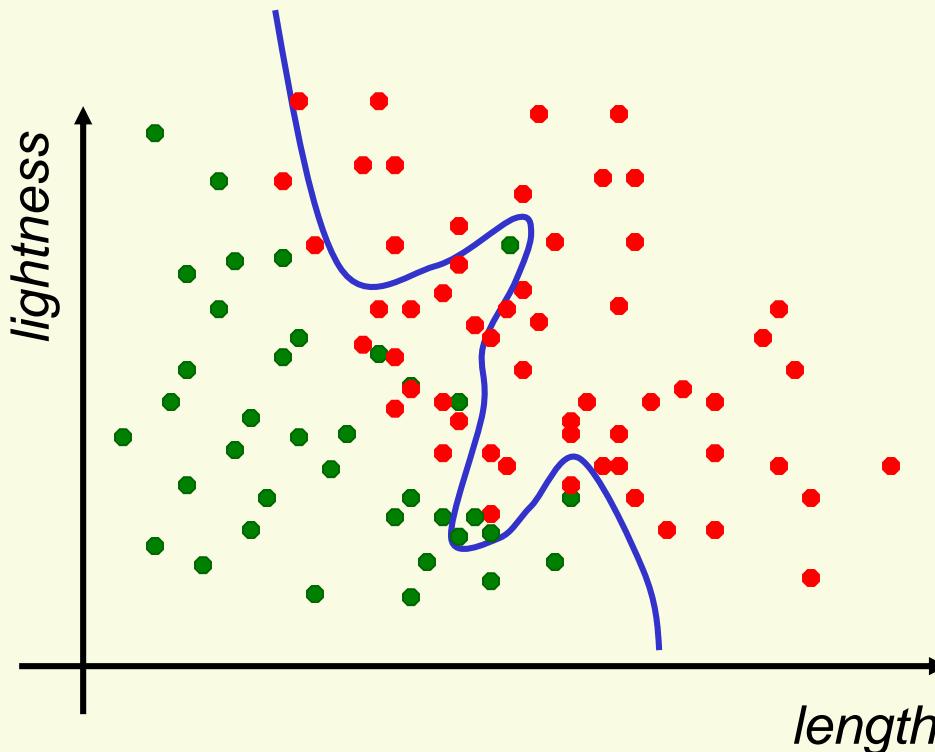
Better decision boundary



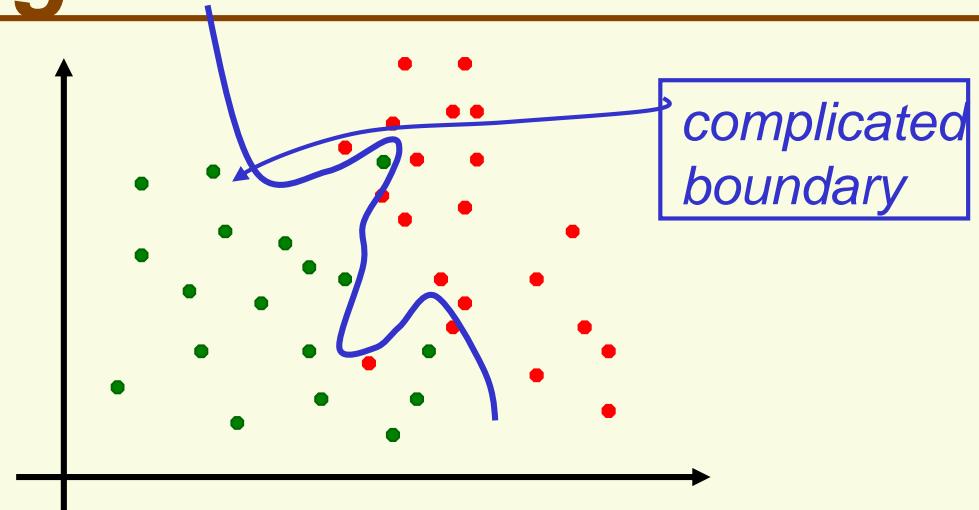
- Ideal decision boundary, 0% classification error

Test Classifier on New Data

- Classifier should perform well on **new** data
- Test “ideal” classifier on new data: **25%** error



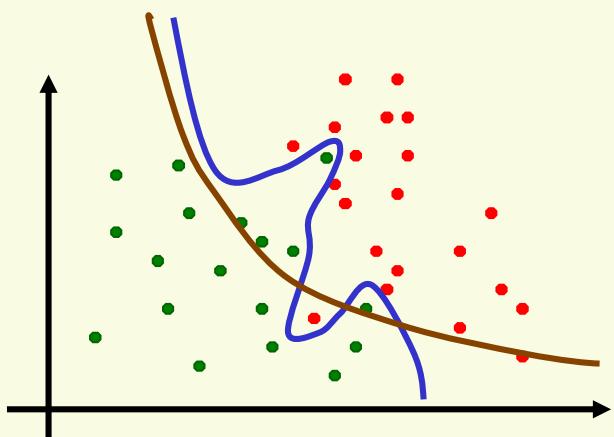
What Went Wrong?



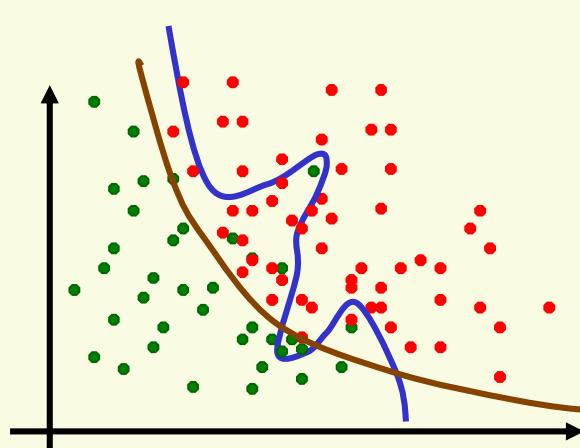
- Poor *generalization*
- Complicated boundaries do not generalize well to the new data, they are too “tuned” to the particular training data, rather than some true model which will separate salmon from sea bass well.
 - This is called overfitting the data

Generalization

training data

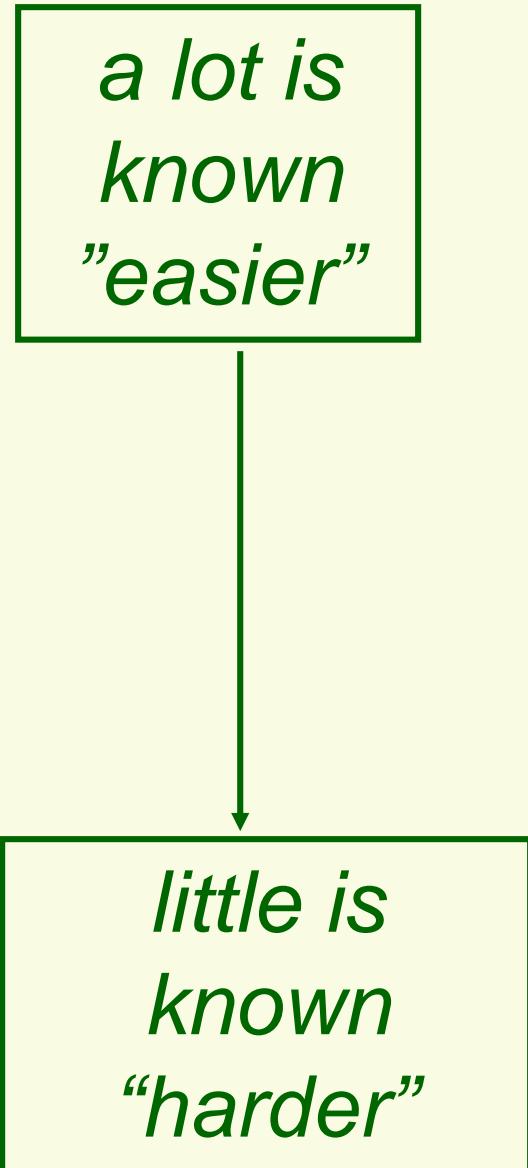


testing data



- Simpler decision boundary does not perform ideally on the training data but generalizes better on new data
- Favor simpler classifiers

Classification Overview

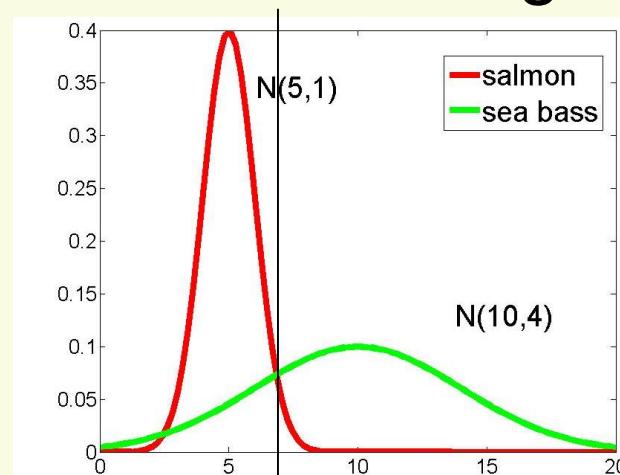


Bayesian Decision theory

- Known probability distribution of the categories
 - never happens in real world
- Do not need training data
- Can design optimal classifier

a lot is known
"easier"

Example
respected fish expert says that salmon's length has distribution $N(5,1)$ and sea bass's length has distribution $N(10,4)$



little is known
"harder"

ML and Bayesian parameter estimation

- Shape of probability distribution is known
 - Happens sometimes
- Labeled training data
- Need to estimate parameters of probability distribution from the training data

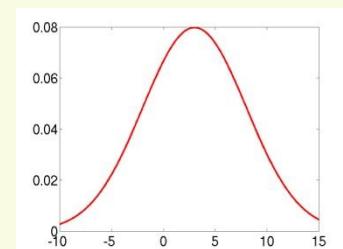


a lot is known "easier"

Example

respected fish expert says salmon's length has distribution $N(\mu_1, \sigma_1^2)$ and sea bass's length has distribution $N(\mu_2, \sigma_2^2)$

- Need to estimate parameters $\mu_1, \sigma_1^2, \mu_2, \sigma_2^2$
- Then can use the methods from the Bayesian decision theory

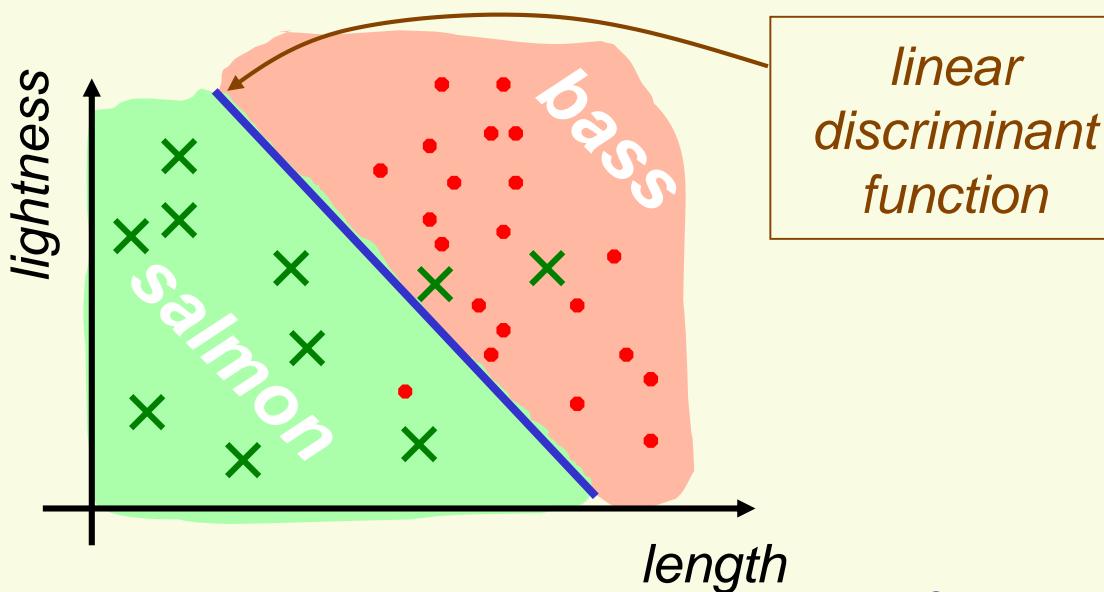


little is known "harder"

Linear discriminant functions

- No probability distribution (no shape or parameters are known)
- Labeled data 
- The shape of discriminant functions is known

a lot is known



- Need to estimate parameters of the discriminant function (parameters of the line in case of linear discriminant)

little is known

Non-Parametric Methods

- Neither probability distribution nor discriminant function is known
 - Happens quite often
- All we have is labeled data



- Estimate the probability distribution from the labeled data

a lot is known "easier"

little is known "harder"

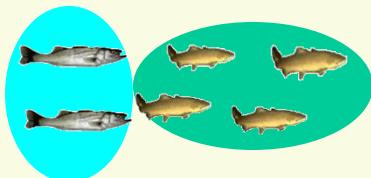
Unsupervised Learning and Clustering

- Data is *not labeled*
 - Happens quite often



*a lot is known
"easier"*

1. Estimate the probability distribution from the *unlabeled* data
2. Cluster the data



*little is known
"harder"*

Classification Summary

1. Bayesian Decision theory (rare case)

- Know probability distribution of the categories
- Do not even need training data
- Can design optimal classifier

a lot is known

2. ML and Bayesian parameter estimation

- Need to estimate Parameters of probability dist.
- Need training data

3. Linear discriminant functions and Neural Nets

- The shape of discriminant functions is known
- Need to estimate parameters of discriminant functions

4. Non-Parametric Methods

- No probability distribution, labeled data

5. Unsupervised Learning and Clustering

- No probability distribution and unlabeled data

little is known

Bayesian Decision Theory

Bayesian Decision Theory

- Know probability distribution of the categories
 - Almost never the case in real life!
 - Nevertheless useful since other cases can be reduced to this one after some work
- Do not even need training data
- Can design optimal classifier

Bayesian Decision theory

Fish Example:

- Each fish is in one of 2 states: sea bass or salmon
- Let ω denote the ***state of nature***
 - $\omega = \omega_1$, for sea bass
 - $\omega = \omega_2$ for salmon
- The state of nature is unpredictable ω is a variable that must be described probabilistically.
 - If the catch produced as much salmon as sea bass the next fish is equally likely to be sea bass or salmon.
- Define:
 - $P(\omega_1)$: **a priori** probability that the next fish is sea bass
 - $P(\omega_2)$: **a priori** probability that the next fish is salmon.

Bayesian Decision theory

- If other types of fish are irrelevant:

$$P(\omega_1) + P(\omega_2) = 1.$$

*Prior probabilities reflect our prior knowledge
(e.g. time of year, fishing area, ...)*

- *Simple decision Rule:*

- *Make a decision without seeing the fish.*
- *Decide ω_1 if $P(\omega_1) > P(\omega_2)$; ω_2 otherwise.*
- *OK if deciding for one fish*
- *If several fish, all assigned to same class*

In general, we have some features and more information.

Cats and Dogs

- Suppose we have these conditional probability mass functions for cats and dogs
 - $P(\text{small ears} \mid \text{dog}) = 0.1$, $P(\text{large ears} \mid \text{dog}) = 0.9$
 - $P(\text{small ears} \mid \text{cat}) = 0.8$, $P(\text{large ears} \mid \text{cat}) = 0.2$
- Observe an animal with large ears
 - Dog or a cat?
 - Makes sense to say dog because probability of observing large ears in a dog is much larger than probability of observing large ears in a cat
 - $P[\text{large ears} \mid \text{dog}] = 0.9 > 0.2 = P[\text{large ears} \mid \text{cat}] = 0.2$
 - We choose the event of larger probability, i.e. maximum likelihood event

Example: Fish Sorting

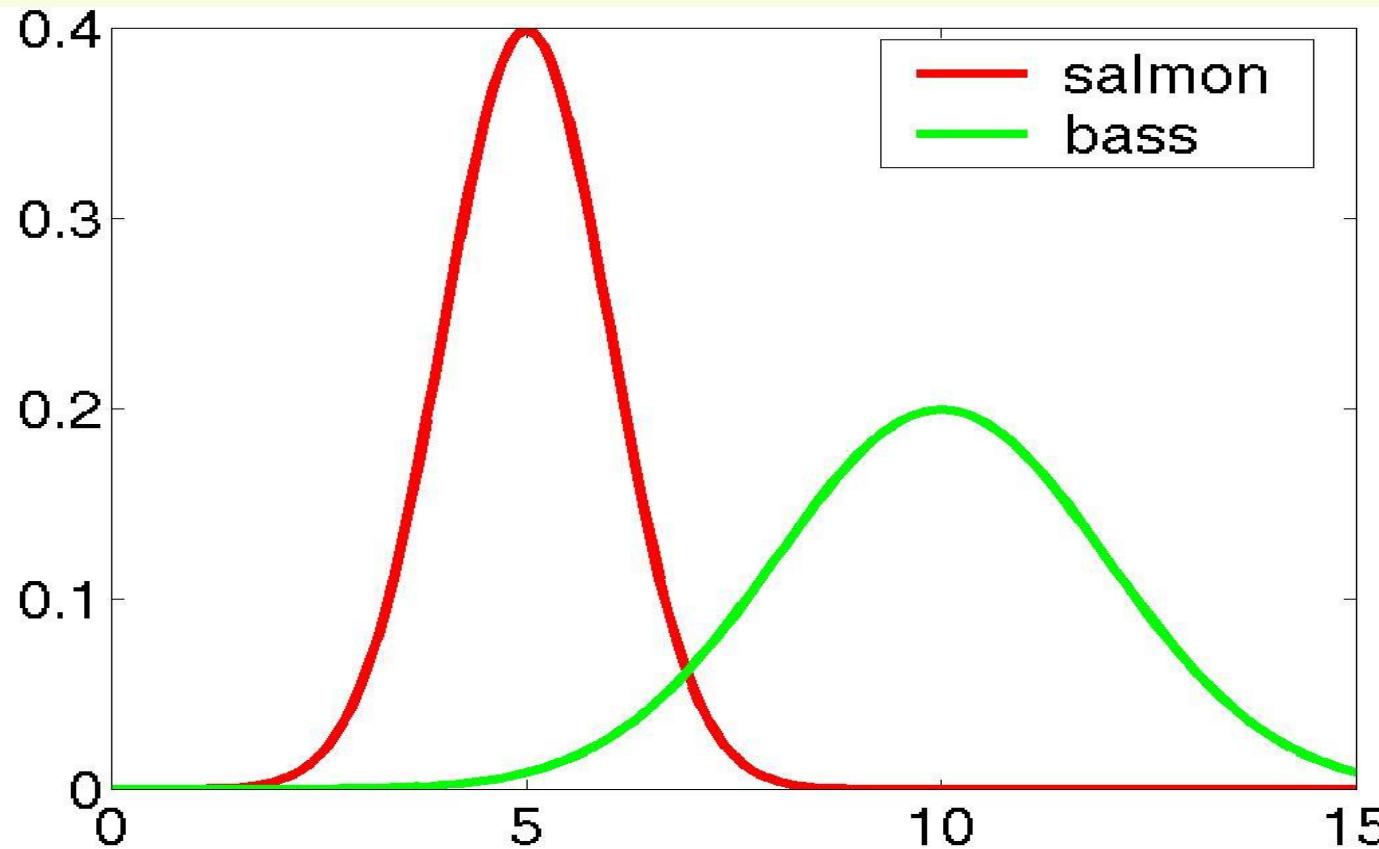
- Respected fish expert says that
 - Salmon's length has distribution $N(5, 1)$
 - Sea bass's length has distribution $N(10, 4)$
- Recall if r.v. is $N(\mu, \sigma^2)$ then it's density is

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Class Conditional Densities

$$p(l \mid \text{salmon})_{\text{fixed}} = \frac{1}{\sqrt{2\pi}} e^{-\frac{(l-5)^2}{2}}$$

$$p(l \mid \text{bass})_{\text{fixed}} = \frac{1}{2\sqrt{2\pi}} e^{-\frac{(l-10)^2}{2^2}}$$



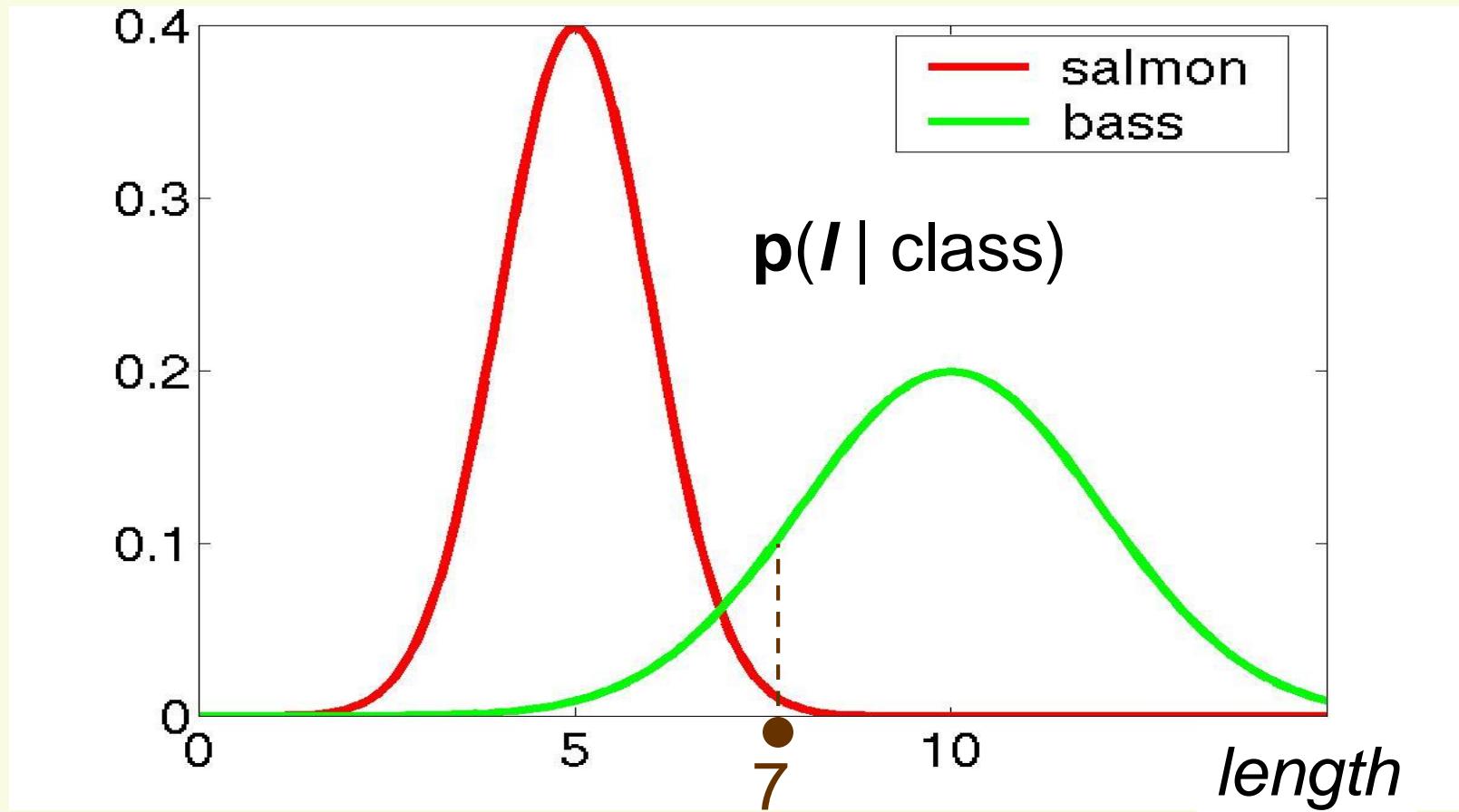
Likelihood function

- Fix length, let fish class vary. Then we get *likelihood function* (it is **not density** and **not probability mass**)

$$p(l | \text{class}) = \begin{cases} \frac{1}{\sqrt{2\pi}} e^{-\frac{(l-5)^2}{2}} & \text{if } \text{class} = \text{salmon} \\ \frac{1}{2\sqrt{2\pi}} e^{-\frac{(l-10)^2}{8}} & \text{if } \text{class} = \text{bass} \end{cases}$$

fixed

Likelihood vs. Class Conditional Density



Suppose a fish has length 7. How do we classify it?

ML (maximum likelihood) Classifier

- We would like to choose salmon if

$$\Pr [\text{length} = 7 | \text{salmon}] > \Pr [\text{length} = 7 | \text{bass}]$$

- However, since ***length*** is a continuous r.v.,

$$\Pr [\text{length} = 7 | \text{salmon}] = \Pr [\text{length} = 7 | \text{bass}] = 0$$

- Instead, we choose class which maximizes likelihood

$$p(l | \text{salmon}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(l-5)^2}{2}}$$

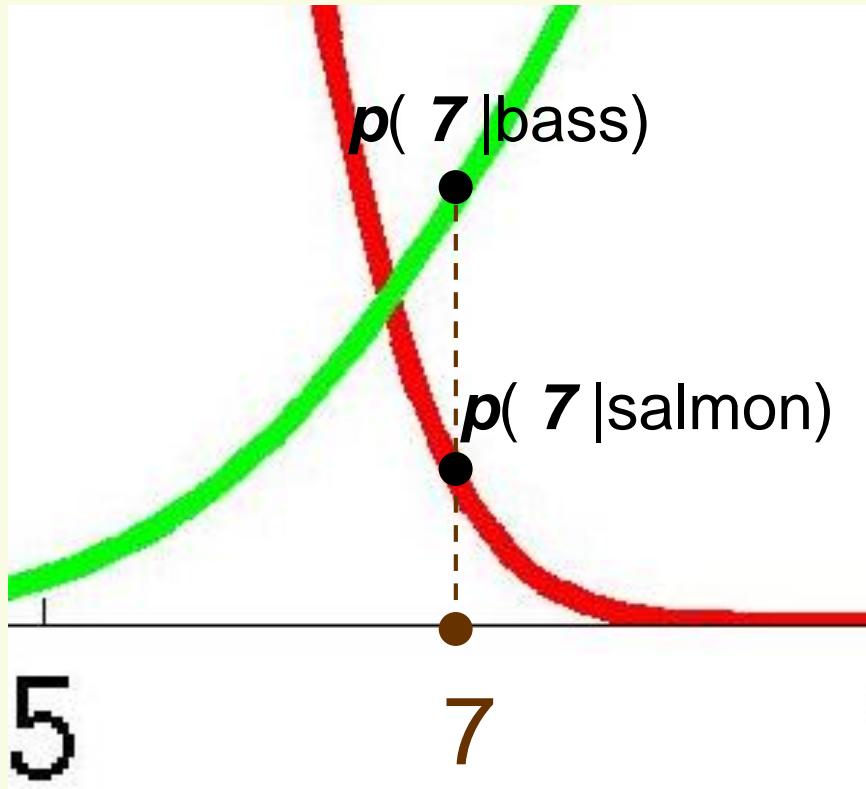
$$p(l | \text{bass}) = \frac{1}{2\sqrt{2\pi}} e^{-\frac{(l-10)^2}{2*4}}$$

- **ML classifier:** for an observed l :

$$\begin{array}{c} \text{bass} < \\ p(l | \text{salmon}) ? p(l | \text{bass}) \\ > \text{salmon} \end{array}$$

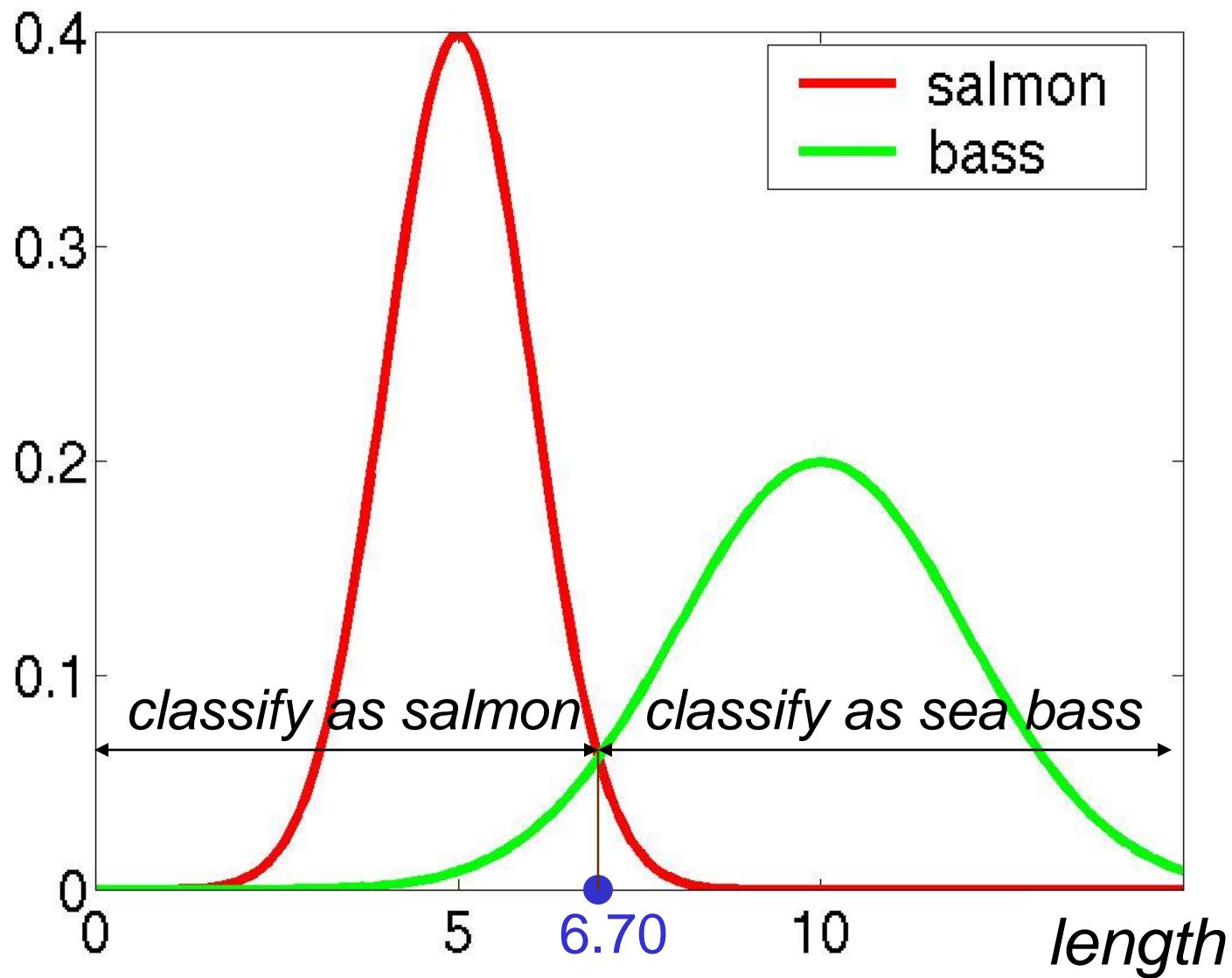
in words: if $p(l | \text{salmon}) > p(l | \text{bass})$,
classify as salmon, else classify as bass

ML (maximum likelihood) Classifier



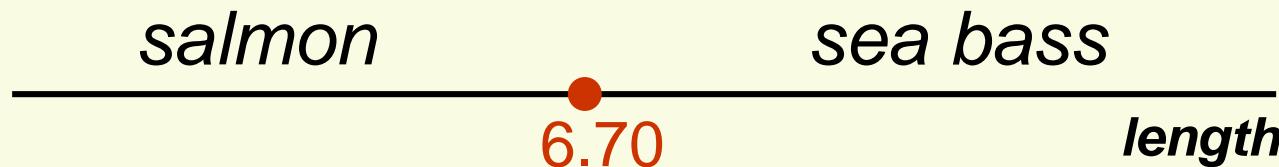
Thus we choose the class (bass) which is more likely to have given the observation

Decision Boundary

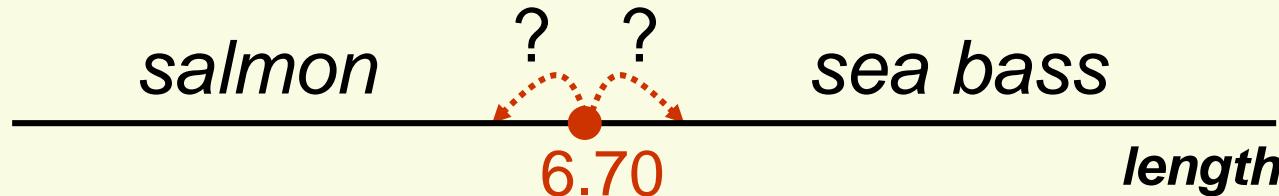


How Prior Changes Decision Boundary?

- Without priors



- How should this change with prior?
 - $P(\text{salmon}) = 2/3$
 - $P(\text{bass}) = 1/3$



Bayes Decision Rule

1. Have likelihood functions
 $p(\text{length} \mid \text{salmon})$ and $p(\text{length} \mid \text{bass})$
 2. Have priors $P(\text{salmon})$ and $P(\text{bass})$
- **Question:** Having observed fish of certain length, do we classify it as salmon or bass?
 - **Natural Idea:**
 - salmon if $P(\text{salmon} \mid \text{length}) > P(\text{bass} \mid \text{length})$
 - bass if $P(\text{bass} \mid \text{length}) > P(\text{salmon} \mid \text{length})$

Posterior

- $P(\text{salmon} \mid \text{length})$ and $P(\text{bass} \mid \text{length})$ are called **posterior** distributions, because the data (length) was revealed (post data)
- How to compute posteriors? Not obvious
- From Bayes rule:

$$P(\text{salmon} \mid \text{length}) = \frac{p(\text{length} \mid \text{salmon}) P(\text{salmon})}{p(\text{length})}$$

- Similarly:

$$P(\text{bass} \mid \text{length}) = \frac{p(\text{length} \mid \text{bass}) P(\text{bass})}{p(\text{length})}$$

MAP (maximum a posteriori) classifier

$$P(\text{salmon} \mid \text{length}) > P(\text{bass} \mid \text{length})$$

bass <

$$\frac{p(\text{length} \mid \text{salmon})P(\text{salmon})}{p(\text{length})} > \frac{p(\text{length} \mid \text{bass})P(\text{bass})}{p(\text{length})}$$

bass <

$$p(\text{length} \mid \text{salmon})P(\text{salmon}) > p(\text{length} \mid \text{bass})P(\text{bass})$$

bass <

Back to Fish Sorting Example

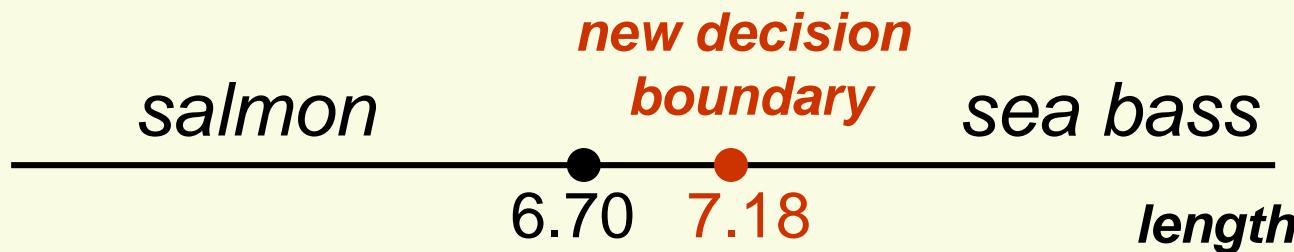
- Likelihood

$$p(l | \text{salmon}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(l-5)^2}{2}}$$

$$p(l | \text{bass}) = \frac{1}{2\sqrt{2\pi}} e^{-\frac{(l-10)^2}{8}}$$

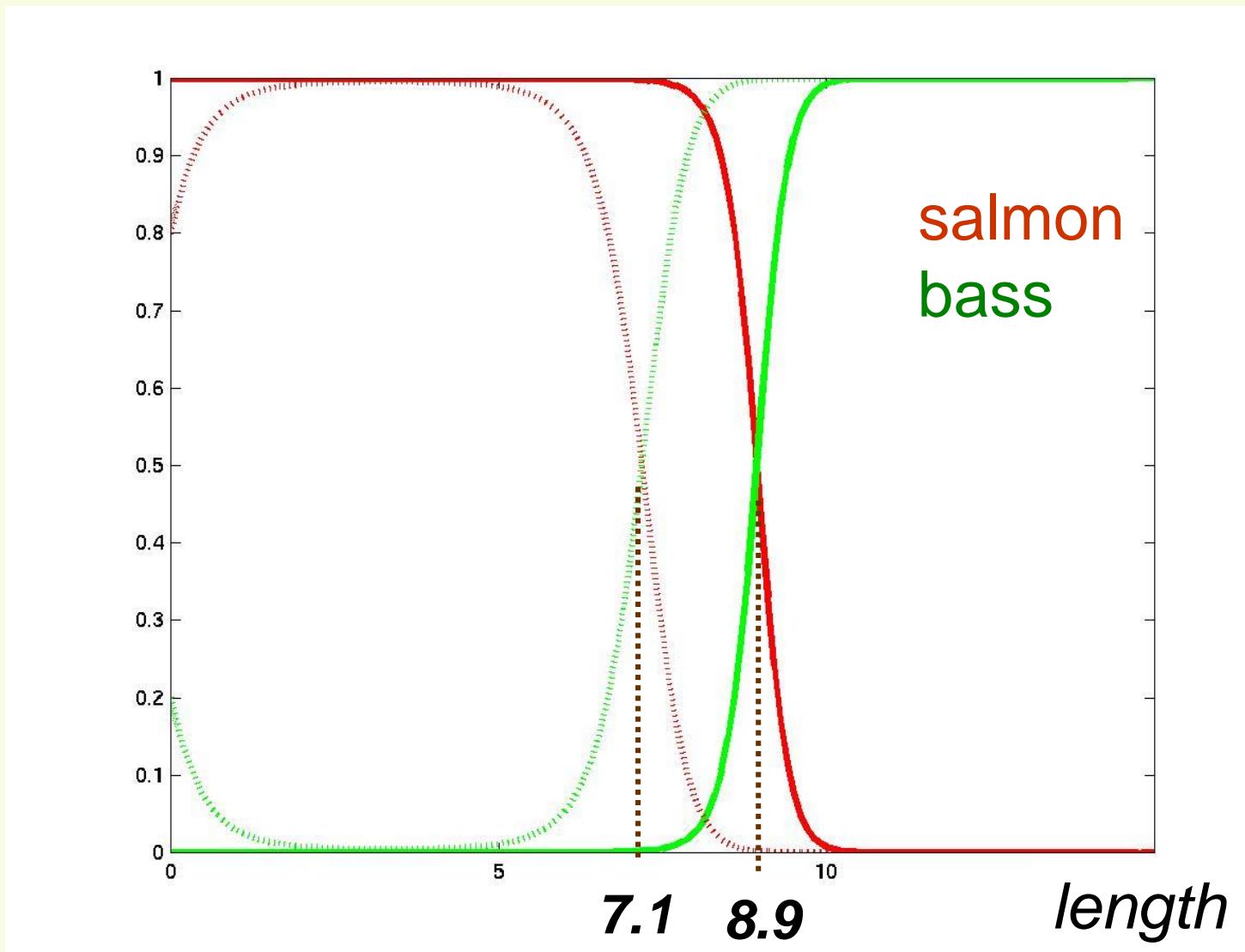
- Priors: $P(\text{salmon}) = 2/3$, $P(\text{bass}) = 1/3$

- Solve inequality $\frac{1}{\sqrt{2\pi}} e^{-\frac{(l-5)^2}{2}} * \frac{2}{3} > \frac{1}{2\sqrt{2\pi}} e^{-\frac{(l-10)^2}{8}} * \frac{1}{3}$

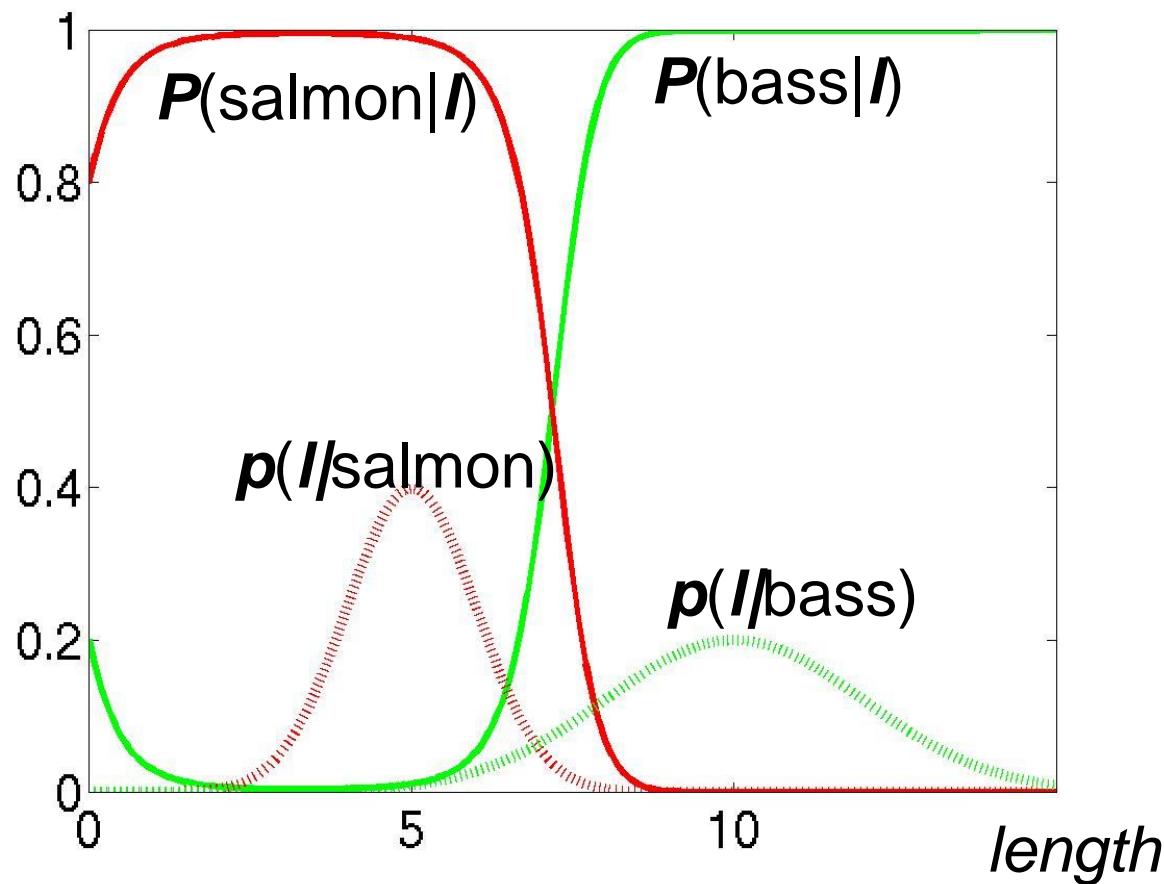


- New decision boundary makes sense since we expect to see more salmon

*Prior $P(s)=2/3$ and $P(b)= 1/3$ vs.
Prior $P(s)=0.999$ and $P(b)= 0.001$*



Likelihood vs Posteriors



likelihood
 $p(I|\text{fish class})$

density with
respect to
length, area
under the
curve is 1

posterior $P(\text{fish class} | I)$

mass function with respect to fish class, so for
each I , $P(\text{salmon} | I) + P(\text{bass} | I) = 1$

More on Posterior

posterior density
(our goal)

$$P(c | I) =$$

likelihood
(given)
 $P(I | c)$

$$P(I)$$

Prior
(given)
 $P(c)$

normalizing factor, often do not even need it for classification since $P(I)$ does not depend on class c . If we do need it, from the law of total probability:

$$P(I) = p(I | \text{salmon})p(\text{salmon}) + p(I | \text{bass})p(\text{bass})$$

Notice this formula consists of likelihoods and priors, which are given

More on Priors

- Prior comes from prior knowledge, no data has been seen yet
- If there is a reliable source prior knowledge, it should be used
- Some problems cannot even be solved reliably without a good prior

More on Map Classifier

$$P(c | I) = \frac{\text{posterior} \quad \text{likelihood} \quad \text{prior}}{P(I)} \frac{P(I | c) P(c)}{P(I)}$$

- Do not care about $P(I)$ when maximizing $P(c|I)$

$$P(c | I) \underset{\propto}{\text{proportional}} P(I | c) P(c)$$

- If $P(\text{salmon})=P(\text{bass})$ (uniform prior) MAP classifier becomes ML classifier $P(c | I) \propto P(I | c)$
- If for some observation I , $P(I|\text{salmon})=P(I|\text{bass})$, then this observation is uninformative and decision is based solely on the prior $P(c | I) \propto P(c)$

Justification for MAP Classifier

- Let's compute probability of error for the MAP estimate:

$$P(\text{salmon} | I) \stackrel{\text{bass}}{<} \stackrel{\text{salmon}}{>} P(\text{bass} | I)$$

- For any particular I , probability of error

$$Pr[\text{error} | I] = \begin{cases} P(\text{bass} | I) & \text{if we decide salmon} \\ P(\text{salmon} | I) & \text{if we decide bass} \end{cases}$$

Thus MAP classifier is optimal for each individual I !

Justification for MAP Classifier

- We are interested to minimize error not just for one I , we really want to minimize the average error over all I

$$Pr[\text{error}] = \int_{-\infty}^{\infty} p(\text{error}, I) dI = \int_{-\infty}^{\infty} Pr[\text{error} | I] p(I) dI$$

- If $Pr[\text{error} | I]$ is as small as possible, the integral is small as possible
- But Bayes rule makes $Pr[\text{error} | I]$ as small as possible

Thus MAP classifier minimizes the probability of error!

More General Case

- Let's generalize a little bit
 - Have more than one feature $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_d]$
 - Have more than 2 classes $\{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_m\}$

More General Case

- As before, for each j we have
 - $p(x/c_j)$ is likelihood of observation x given that the true class is c_j ,
 - $P(c_j)$ is prior probability of class c_j ,
 - $P(c_j/x)$ is posterior probability of class c_j given that we observed data x
- Evidence, or probability density for data

$$p(x) = \sum_{j=1}^m p(x/c_j)P(c_j)$$

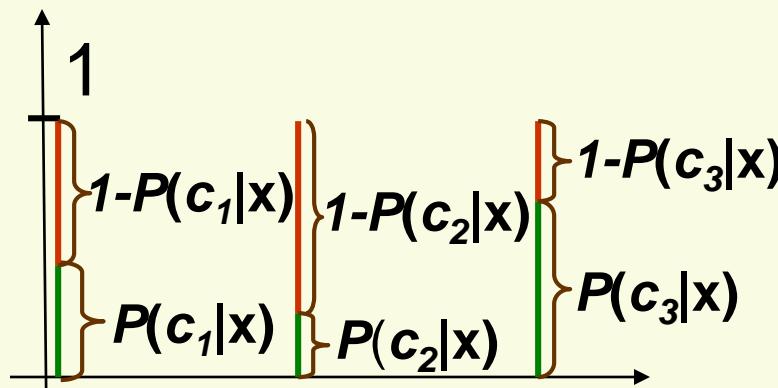
Minimum Error Rate Classification

- Want to minimize average probability of error

$$Pr[\text{error}] = \int p(\text{error}, x) dx = \int Pr[\text{error} | x] p(x) dx$$

*need to make this
as small as possible*

- $Pr[\text{error} | x] = 1 - P(c_i | x)$ if we decide class c_i
 - $Pr[\text{error} | x]$ is minimized with MAP classifier
 - Decide on class c_i if $P(c_i | x) > P(c_j | x) \forall j \neq i$
- MAP classifier is optimal
If we want to minimize the probability of error*



General Bayesian Decision Theory

- In some cases we may want to refuse to make a decision (let human expert handle tough case)
 - allow actions $\{\alpha_1, \alpha_2, \dots, \alpha_k\}$
- Suppose some mistakes are more costly than others (classifying a benign tumor as cancer is not as bad as classifying cancer as benign tumor)
 - Allow loss functions $\lambda(\alpha_i / c_j)$ describing loss occurred when taking action α_i when the true class is c_j

Conditional Risk

- Suppose we observe x and wish to take action α_i ,
- If the true class is c_j , by definition, we incur loss $\lambda(\alpha_i / c_j)$
- Probability that the true class is c_j after observing x is $P(c_j / x)$
- The expected loss associated with taking action α_i is called ***conditional risk*** and it is:

$$R(\alpha_i / x) = \sum_{j=1}^m \lambda(\alpha_i / c_j) P(c_j / x)$$

Conditional Risk

sum over disjoint events
(different classes)

probability of
class \mathbf{c}_j given
observation x

$$R(\alpha_i | x) = \sum_{j=1}^m \underbrace{\lambda(\alpha_i | \mathbf{c}_j)}_{\text{penalty for taking action } \alpha_i \text{ if observe } x} P(\mathbf{c}_j | x)$$

part of overall penalty
which comes from event
that true class is \mathbf{c}_j

Example: Zero-One loss function

- action α_i is decision that true class is c_i ,

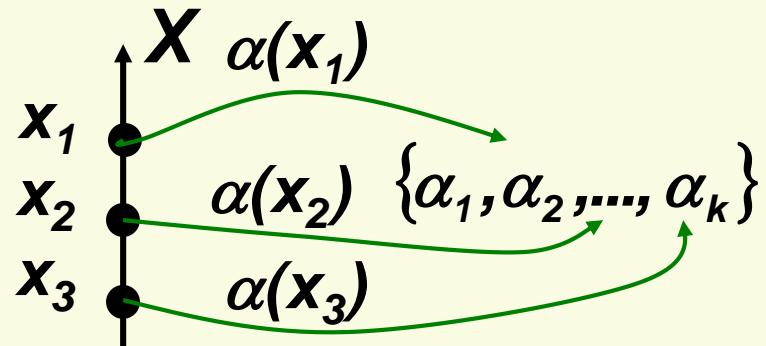
$$\lambda(\alpha_i | c_j) = \begin{cases} 0 & \text{if } i = j \quad (\text{no mistake}) \\ 1 & \text{otherwise} \quad (\text{mistake}) \end{cases}$$

$$\begin{aligned} R(\alpha_i | x) &= \sum_{j=1}^m \lambda(\alpha_i | c_j) P(c_j | x) = \sum_{i \neq j} P(c_j | x) = \\ &= 1 - P(c_i | x) = \Pr[\text{error if decide } c_i] \end{aligned}$$

- Thus MAP classifier optimizes $R(\alpha_i | x)$
 $P(c_i | x) > P(c_j | x) \quad \forall j \neq i$
- MAP classifier is Bayes decision rule under zero-one loss function

Overall Risk

- Decision rule is a function $\alpha(\mathbf{x})$ which for every x specifies action out of $\{\alpha_1, \alpha_2, \dots, \alpha_k\}$



- The average risk for $\alpha(\mathbf{x})$

$$R(\alpha) = \int R(\alpha(\mathbf{x}) / \mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

need to make this as small as possible

- Bayes decision rule $\alpha(\mathbf{x})$ for every x is the action which minimizes the conditional risk

$$R(\alpha_i / \mathbf{x}) = \sum_{j=1}^m \lambda(\alpha_i / c_j) P(c_j / \mathbf{x})$$

- Bayes decision rule $\alpha(\mathbf{x})$ is **optimal**, i.e. gives the minimum possible overall risk R^*

Bayes Risk: Example

- Salmon is more tasty and expensive than sea bass

$$\lambda_{sb} = \lambda(\text{salmon} | \text{bass}) = 2 \quad \text{classify bass as salmon}$$

$$\lambda_{bs} = \lambda(\text{bass} | \text{salmon}) = 1 \quad \text{classify salmon as bass}$$

$$\lambda_{ss} = \lambda_{bb} = 0 \quad \text{no mistake, no loss}$$

- Likelihoods $p(I | \text{salmon}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(I-5)^2}{2}}$ $p(I | \text{bass}) = \frac{1}{2\sqrt{2\pi}} e^{-\frac{(I-10)^2}{2^2}}$

- Priors $P(\text{salmon}) = P(\text{bass})$

- Risk $R(\alpha | x) = \sum_{j=1}^m \lambda(\alpha | c_j) P(c_j | x) = \lambda_{\alpha s} P(s | I) + \lambda_{\alpha b} P(b | I)$

$$R(\text{salmon} | I) = \lambda_{ss} P(s | I) + \lambda_{sb} P(b | I) = \lambda_{sb} P(b | I)$$

$$R(\text{bass} | I) = \lambda_{bs} P(s | I) + \lambda_{bb} P(b | I) = \lambda_{bs} P(s | I)$$

Bayes Risk: Example

$$R(\text{salmon} | I) = \lambda_{sb} P(b | I) \quad R(\text{bass} | I) = \lambda_{bs} P(s | I)$$

- Bayes decision rule (optimal for our loss function)

$$\lambda_{sb} P(b | I) \stackrel{\text{^{salmon}}}{?} \lambda_{bs} P(s | I)$$

$\begin{matrix} < \\ > \end{matrix}$ **bass**

- Need to solve $\frac{P(b | I)}{P(s | I)} < \frac{\lambda_{bs}}{\lambda_{sb}}$
- Or, equivalently, since priors are equal:

$$\frac{P(I | b) P(b) p(I)}{p(I) P(I | s) P(s)} = \frac{P(I | b)}{P(I | s)} < \frac{\lambda_{bs}}{\lambda_{sb}}$$

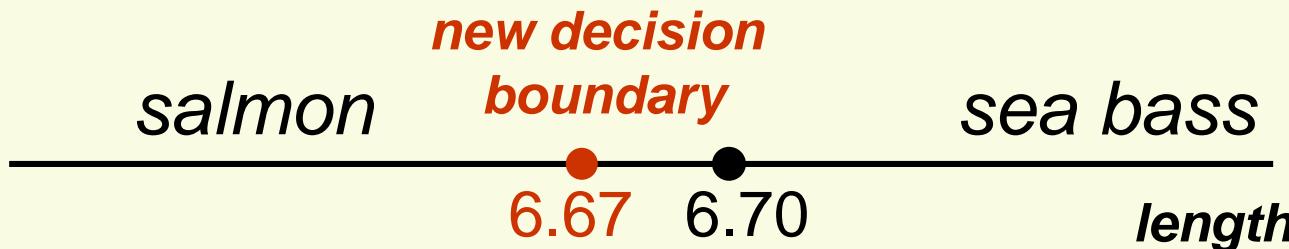
Bayes Risk: Example

- Need to solve $\frac{P(I|b)}{P(I|s)} < \frac{\lambda_{bs}}{\lambda_{sb}}$

- Substituting likelihoods and losses

$$\frac{2 \cdot \sqrt{2\pi} \exp^{-\frac{(I-10)^2}{8}}}{1 \cdot 2 \sqrt{2\pi} \exp^{-\frac{(I-5)^2}{2}}} < 1 \Leftrightarrow \frac{\exp^{-\frac{(I-10)^2}{8}}}{\exp^{-\frac{(I-5)^2}{2}}} < 1 \Leftrightarrow \ln\left(\frac{\exp^{-\frac{(I-10)^2}{8}}}{\exp^{-\frac{(I-5)^2}{2}}}\right) < \ln(1) \Leftrightarrow$$

$$\Leftrightarrow -\frac{(I-10)^2}{8} + \frac{(I-5)^2}{2} < 0 \Leftrightarrow 3I^2 - 20I < 0 \Leftrightarrow 0 \leq I < 6.6667$$



Likelihood Ratio Rule

- In 2 category case, use likelihood ratio rule

$$\frac{P(x | c_1)}{P(x | c_2)} > \frac{\lambda_{12} - \lambda_{22}}{\lambda_{21} - \lambda_{11}} \frac{P(c_2)}{P(c_1)}$$

*likelihood
ratio* *fixed number
Independent of x*

- If above inequality holds, decide c_1
- Otherwise decide c_2

Discriminant Functions

- All decision rules have the same structure:
at observation x choose class c_i , s.t.

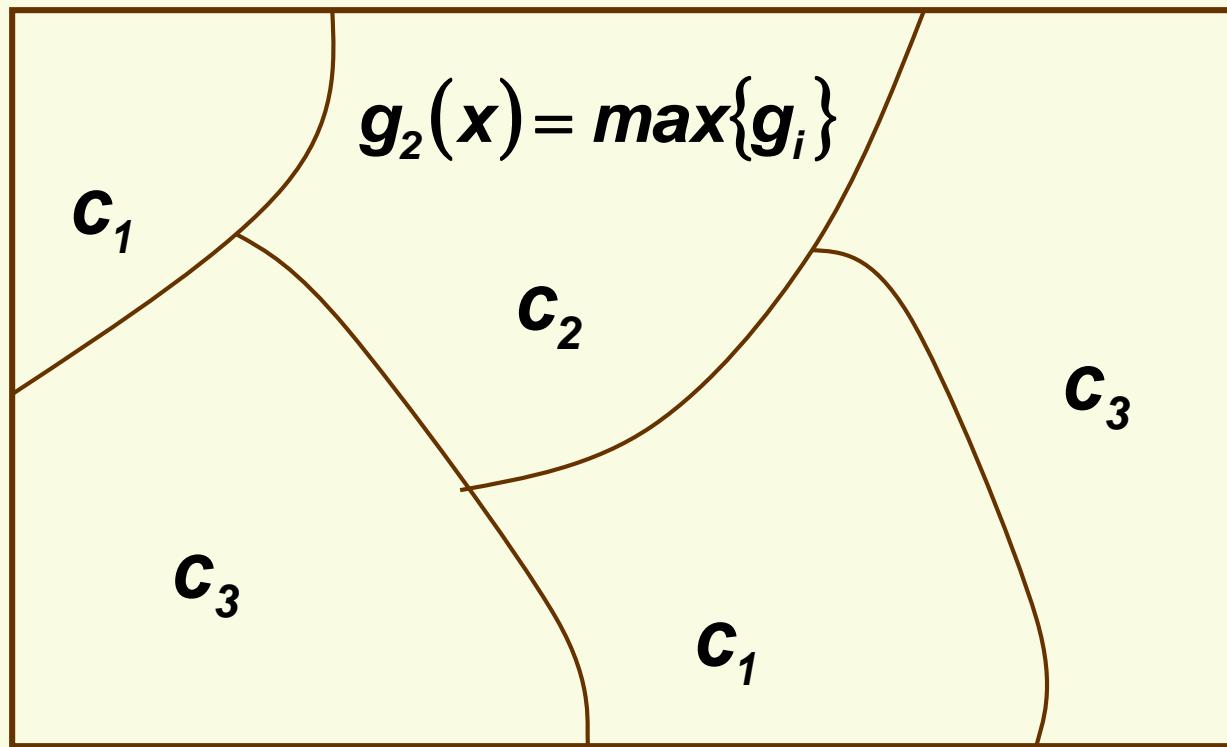
$$g_i(x) > g_j(x) \quad \forall j \neq i$$

*discriminant
function*

- ML decision rule: $g_i(x) = P(x | c_i)$
- MAP decision rule: $g_i(x) = P(c_i | x)$
- Bayes decision rule: $g_i(x) = -R(c_i | x)$

Decision Regions

- Discriminant functions split the feature vector space X into decision regions



Important Points

- If we know probability distributions for the classes, we can design the **optimal classifier**
- Definition of “optimal” depends on the chosen loss function
 - Under the minimum error rate (zero-one loss function)
 - No prior: ML classifier is optimal
 - Have prior: MAP classifier is optimal
 - More general loss function
 - General Bayes classifier is optimal

Bayesian Decision Theory Tutorial

Tutorial 1 – the outline

- ◆ Bayesian decision making with discrete probabilities – an example
- ◆ Looking at continuous densities
- ◆ Bayesian decision making with continuous probabilities – an example
- ◆ The Bayesian Doctor Example

Example 1 – checking on a course

- ◆ A student needs to achieve a decision on which courses to take, based only on his first lecture.
- ◆ Define 3 categories of courses ω_j : **good, fair, bad**.
- ◆ From his previous experience, he knows:

Quality of the course	good	fair	bad
Probability (prior)	0.2	0.4	0.4

- ◆ These are **prior probabilities**.

Example 1 – continued

- The student also knows the **class-conditionals**:

$\Pr(x \omega_j)$	good	fair	bad
Interesting lecture	0.8	0.5	0.1
Boring lecture	0.2	0.5	0.9

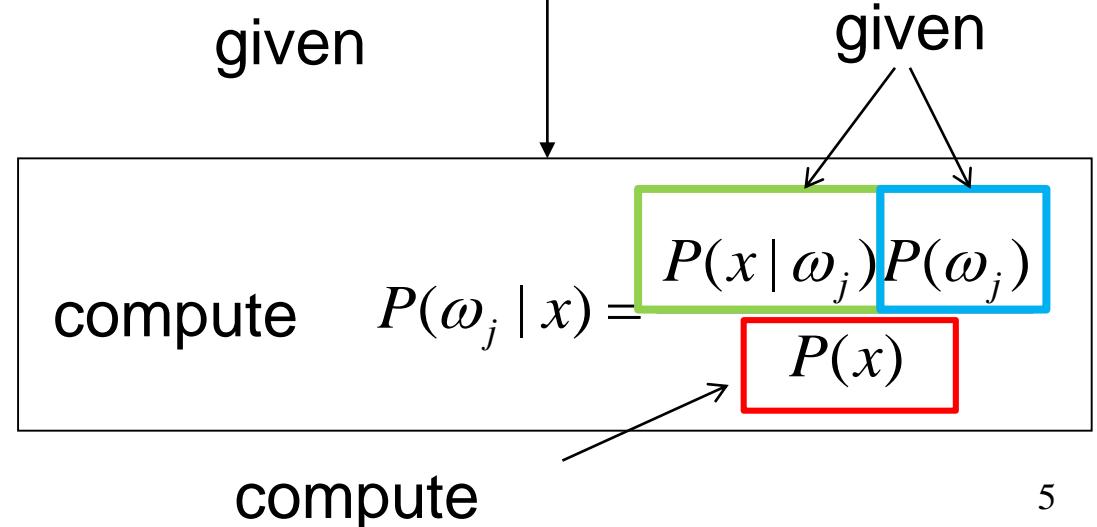
- The **loss function** is given by the matrix

$\lambda(a_i \omega_j)$	good course	fair course	bad course
Taking the course	0	5	10
Not taking the course	20	5	0

Example 1 – continued

- The student wants to make an optimal decision=> minimal possible $R(\alpha)$,
while $\alpha: x \rightarrow \{\text{take the course}, \text{drop the course}\}$
- The student needs to minimize the conditional risk;

$$R(\alpha_i | x) = \sum_{j=1}^c \lambda(\alpha_i | \omega_j) P(\omega_j | x)$$



Example 1 : compute $P(x)$

- The probability to get an “interesting lecture” ($x = \text{interesting}$):

$$\begin{aligned}\Pr(\text{interesting}) &= \Pr(\text{interesting}|\text{good course}) * \Pr(\text{good course}) \\ &\quad + \Pr(\text{interesting}|\text{fair course}) * \Pr(\text{fair course}) \\ &\quad + \Pr(\text{interesting}|\text{bad course}) * \Pr(\text{bad course}) \\ &= 0.8 * 0.2 + 0.5 * 0.4 + 0.1 * 0.4 = 0.4\end{aligned}$$

- Consequently, $\Pr(\text{boring}) = 1 - 0.4 = 0.6$

Example 1 : compute $P(\omega_j \mid x)$

Suppose the lecture was **interesting**. Then we want to compute the **posterior** probabilities of each one of the 3 possible “states of nature”.

$$\Pr(\text{good course} \mid \text{interesting lecture})$$

$$= \frac{\Pr(\text{interesting} \mid \text{good})\Pr(\text{good})}{\Pr(\text{interesting})} = \frac{0.8 * 0.2}{0.4} = 0.4$$

$$\Pr(\text{fair} \mid \text{interesting})$$

$$= \frac{\Pr(\text{interesting} \mid \text{fair})\Pr(\text{fair})}{\Pr(\text{interesting})} = \frac{0.5 * 0.4}{0.4} = 0.5$$

- We can get $\Pr(\text{bad} \mid \text{interesting})=0.1$ either by the same method, or by noting that it complements to 1 the above two.

Example 1

$$R(\alpha_i | x) = \sum_{j=1}^c \lambda(\alpha_i | \omega_j) P(\omega_j | x)$$

- The student needs to minimize the conditional risk; take the course:

$$\begin{aligned} R(\text{taking} | \text{interesting}) &= \lambda(\text{taking} | \text{good}) \Pr(\text{good} | \text{interesting}) \\ &\quad + \lambda(\text{taking} | \text{fair}) \Pr(\text{fair} | \text{interesting}) \\ &\quad + \lambda(\text{taking} | \text{bad}) \Pr(\text{bad} | \text{interesting}) \\ &= 0.4*0+0.5*5+0.1*10=3.5 \end{aligned}$$

or drop it:

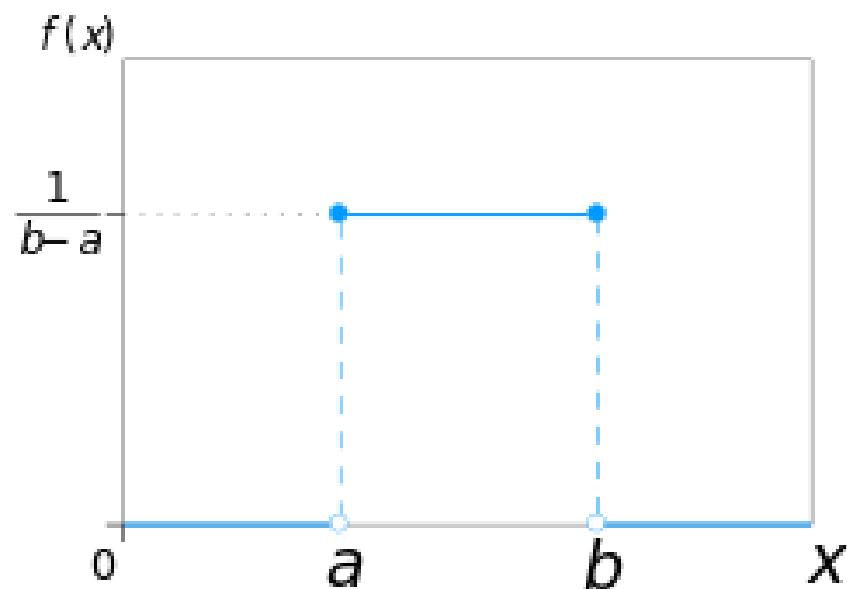
$$\begin{aligned} R(\text{droping} | \text{interesting}) &= \lambda(\text{droping} | \text{good}) \Pr(\text{good} | \text{interesting}) \\ &\quad + \lambda(\text{droping} | \text{fair}) \Pr(\text{fair} | \text{interesting}) \\ &\quad + \lambda(\text{droping} | \text{bad}) \Pr(\text{bad} | \text{interesting}) \\ &= 0.4*20+0.5*5+0.1*0=10.5 \end{aligned}$$

Constructing an optimal decision function

- So, if the first lecture was interesting, the student will minimize the conditional risk by taking the course.
- In order to construct the full decision function, we need to define the risk minimization action for the case of boring lecture, as well.

Do it!

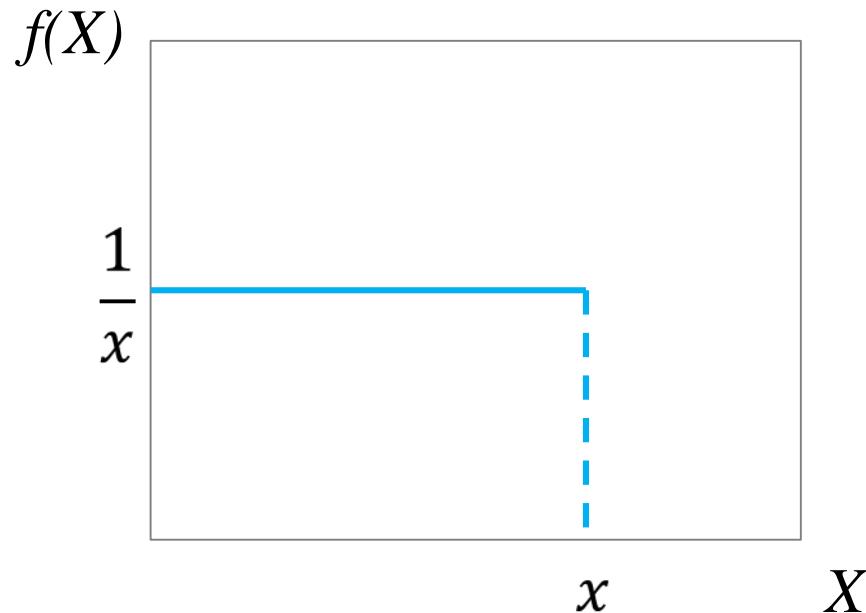
Uniform Distribution



Example 2 – continuous density

- Let X be a real value r.v., representing a number randomly picked from the interval $[0,1]$; its distribution is known to be uniform.
- Then let Y be a real r.v. whose value is chosen at random from $[0, X]$ also with uniform distribution.
- We are presented with the value of Y , and need to “guess” the most “likely” value of X .
- In a more formal fashion: given the value of Y , find the probability density function of X and determine its maxima.

Uniform Distribution



Example 2 – continued

- What we look for is $P(X=x | Y=y)$ – that is, the **p.d.f.**
- The class-conditional (given the value of X):

$$P(Y = y | X = x) = \begin{cases} \frac{1}{x} & y \leq x \leq 1 \\ 0 & y > x \end{cases}$$

- For the given evidence:

$$P(Y = y) = \int_y^1 \frac{1}{x} dx = \ln\left(\frac{1}{y}\right)$$

(using total probability)

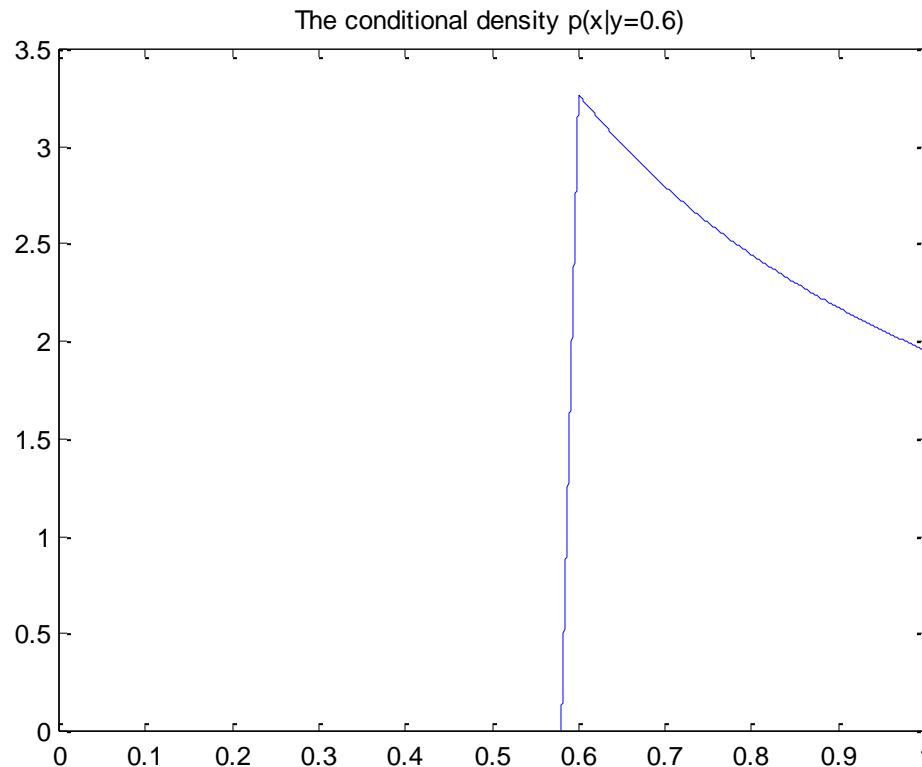
Example 2 – conclusion

- Applying Bayes' rule:

$$p(x | y) = \frac{p(y | x)p(x)}{p(y)} = \frac{\frac{1}{x} - 1}{\ln\left(\frac{1}{y}\right)}$$

- This is monotonically decreasing function, over $[y, 1]$.
- So (informally) the most “likely” value of X (the one with highest probability density value) is $X=y$.

Illustration – conditional p.d.f.



Example 3: hiring a secretary

- A manager needs to hire a new secretary, and a good one.

- Unfortunately, good secretary are hard to find:

$$\Pr(w_g) = 0.2, \quad \Pr(w_b) = 0.8$$

- The manager decides to use a new test. The grade is a real number in the range from 0 to 100.
- The manager's estimation of the possible losses:

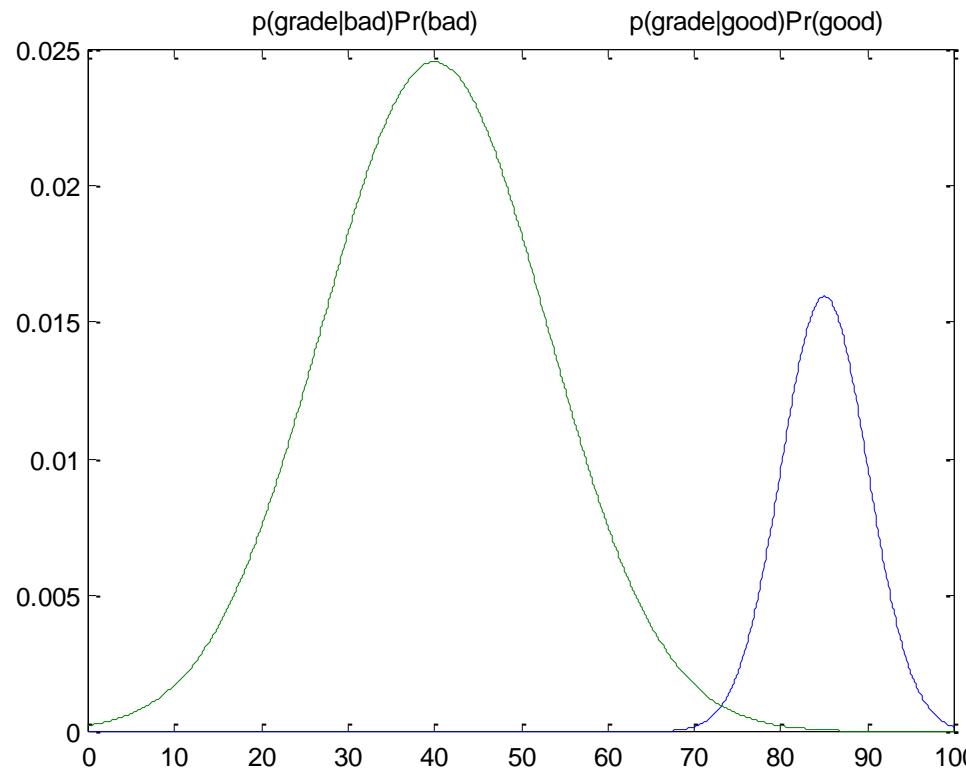
$\lambda(\text{decision}, w_i)$	w_g	w_b
Hire	0	20
Reject	5	0

Example 3: continued

- The class conditional densities are known to be approximated by a normal p.d.f.:

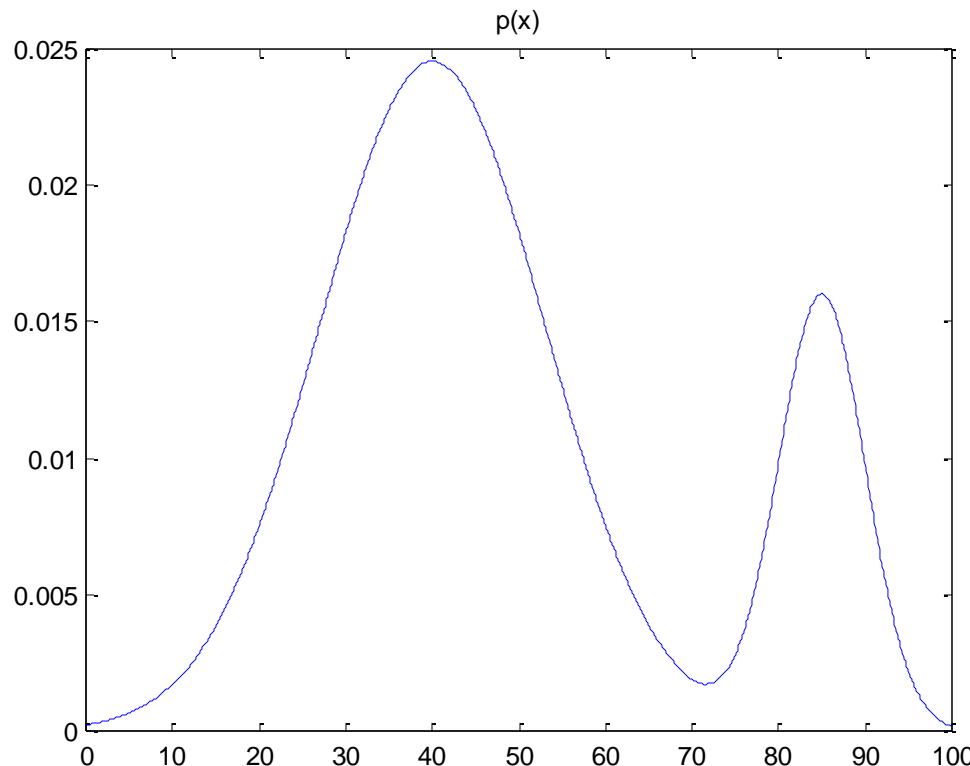
$$p(\text{grade} | \text{good secretary}) \sim N(85, 5)$$

$$p(\text{grade} | \text{bad secretary}) \sim N(40, 13)$$



Example 3: continued

- The resulting probability density for the grade looks as follows: $p(x) = p(x|w_b)p(w_b) + p(x|w_g)p(w_g)$



Example 3: continued

$$R(\alpha_i | x) = \sum_{j=1}^c \lambda(\alpha_i | \omega_j) P(\omega_j | x)$$

- ◆ We need to know for which grade values hiring the secretary would minimize the risk:

$$R(\text{hire} | x) < R(\text{reject} | x) \Leftrightarrow$$

$$p(w_b | x)\lambda(\text{hire}, w_b) + p(w_g | x)\lambda(\text{hire}, w_g)$$

$$< p(w_b | x)\lambda(\text{reject}, w_b) + p(w_g | x)\lambda(\text{reject}, w_g) \Leftrightarrow$$

$$[\lambda(\text{hire}, w_b) - \lambda(\text{reject}, w_b)] \cdot p(w_b | x) < [\lambda(\text{reject}, w_g) - \lambda(\text{hire}, w_g)] p(w_g | x)$$

- ◆ The posteriors are given by

$$p(w_i | x) = \frac{p(x | w_i) p(w_i)}{p(x)}$$

Example 3: continued

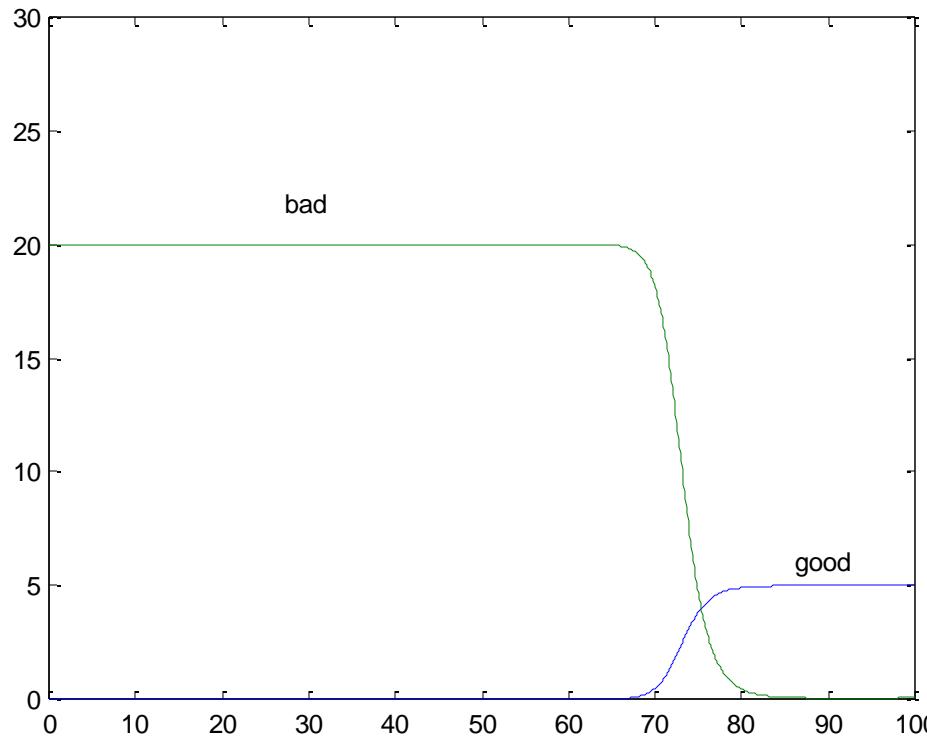
- The posteriors scaled by the loss differences,

$$[\lambda(\text{hire}, w_b) - \lambda(\text{reject}, w_b)] \cdot p(w_b | x)$$

and

$$[\lambda(\text{reject}, w_g) - \lambda(\text{hire}, w_g)] \cdot p(w_g | x)$$

look like:



Example 3: continued

- Numerically, we have:

$$p(x) = \frac{0.2}{5\sqrt{2\pi}} e^{-\frac{(x-85)^2}{2.5^2}} + \frac{0.8}{13\sqrt{2\pi}} e^{-\frac{(x-40)^2}{2.13^2}}$$

$$p(w_b | x) = \frac{\frac{0.8}{13\sqrt{2\pi}} e^{-\frac{(x-40)^2}{2.13^2}}}{p(x)}, \quad p(w_g | x) = \frac{\frac{0.2}{5\sqrt{2\pi}} e^{-\frac{(x-85)^2}{2.5^2}}}{p(x)}$$

- We need to solve $20p(w_b | x) > 5p(w_g | x)$
- Solving numerically yields one solution in $[0, 100]$:

$$x \approx 76$$

The Bayesian Doctor Example

A person doesn't feel well and goes to a doctor.

Assume two states of nature:

ω_1 : The person has a common flue.

ω_2 : The person is really sick (a vicious bacterial infection).

The doctor's **prior** is: $p(\omega_1) = 0.9$ $p(\omega_2) = 0.1$

This doctor has two possible actions: "prescribe" hot tea or antibiotics. Doctor can use prior and predict optimally: always flue. Therefore doctor will always prescribe hot tea.

The Bayesian Doctor - Cntd.

- But there is very high risk: Although this doctor can diagnose with very high rate of success using the prior, (s)he can lose a patient once in a while.
- Denote the two possible actions:
 - a_1 = prescribe hot tea
 - a_2 = prescribe antibiotics
- Now assume the following cost (loss) matrix:

	flue	bacteria
ω_1	ω_1	ω_2
a_1	0	10
a_2	1	0

The Bayesian Doctor - Cntd.

- Choosing a_1 results in **expected risk** of

$$R(a_1) = p(\omega_1) \cdot \lambda_{1,1} + p(\omega_2) \cdot \lambda_{1,2}$$

$$= 0 + 0.1 \cdot 10 = 1$$

- Choosing a_2 results in expected risk of

$$R(a_2) = p(\omega_1) \cdot \lambda_{2,1} + p(\omega_2) \cdot \lambda_{2,2}$$

$$= 0.9 \cdot 1 + 0 = 0.9$$

- So, considering the costs it's much better (and optimal!) to always give antibiotics.

The Bayesian Doctor - Cntd.

- But doctors can do more. For example, they can take some ***observations***.
- A reasonable observation is to perform a blood test.
- Suppose the possible results of the blood test are:
 - x_1 = negative (no bacterial infection)
 - x_2 = positive (infection)
- But blood tests can often fail. Suppose
(*class conditional* probabilities.)

infection $p(x_1 | \omega_2) = 0.3$ $p(x_2 | \omega_2) = 0.7$

flue $p(x_2 | \omega_1) = 0.2$ $p(x_1 | \omega_1) = 0.8$

The Bayesian Doctor - Cntd.

- Define the **conditional risk** given the observation

$$R(a_i | x) = \sum_{\omega_j} p(\omega_j | x) \cdot \lambda_{i,j}$$

- We would like to compute the conditional risk for each action and observation so that the doctor can choose an optimal action that minimizes risk.
- How can we compute $p(\omega_j | x)$?
- We use the class conditional probabilities and **Bayes inversion rule**.

The Bayesian Doctor - Cntd.

- Let's calculate first $p(x_1)$ and $p(x_2)$

$$\begin{aligned} p(x_1) &= p(x_1 | \omega_1) \cdot p(\omega_1) + p(x_1 | \omega_2) \cdot p(\omega_2) \\ &= 0.8 \cdot 0.9 + 0.3 \cdot 0.1 \\ &= 0.75 \end{aligned}$$

- $p(x_2)$ is complementary to $p(x_1)$, so $p(x_2) = 0.25$

The Bayesian Doctor - Cntd.

$$\begin{aligned} R(\alpha_1 \mid x_1) &= p(\omega_1 \mid x_1) \cdot \lambda_{1,1} + p(\omega_2 \mid x_1) \cdot \lambda_{1,2} \\ &= 0 + p(\omega_2 \mid x_1) \cdot 10 \\ &= 10 \cdot \frac{p(x_1 \mid \omega_2) \cdot p(\omega_2)}{p(x_1)} \\ &= 10 \cdot \frac{0.3 \cdot 0.1}{0.75} = 0.4 \end{aligned}$$

$$\begin{aligned} R(\alpha_2 \mid x_1) &= p(\omega_1 \mid x_1) \cdot \lambda_{2,1} + p(\omega_2 \mid x_1) \cdot \lambda_{2,2} \\ &= p(\omega_1 \mid x_1) \cdot 1 + p(\omega_2 \mid x_1) \cdot 0 \\ &= \frac{p(x_1 \mid \omega_1) \cdot p(\omega_1)}{p(x_1)} \\ &= \frac{0.8 \cdot 0.9}{0.75} = 0.96 \end{aligned}$$

The Bayesian Doctor - Cntd.

$$\begin{aligned} R(\alpha_1 \mid x_2) &= p(\omega_1 \mid x_2) \cdot \lambda_{1,1} + p(\omega_2 \mid x_2) \cdot \lambda_{1,2} \\ &= 0 + p(\omega_2 \mid x_2) \cdot 10 \\ &= 10 \cdot \frac{p(x_2 \mid \omega_2) \cdot p(\omega_2)}{p(x_2)} \\ &= 10 \cdot \frac{0.7 \cdot 0.1}{0.25} = 2.8 \end{aligned}$$

$$\begin{aligned} R(\alpha_2 \mid x_2) &= p(\omega_1 \mid x_2) \cdot \lambda_{2,1} + p(\omega_2 \mid x_2) \cdot \lambda_{2,2} \\ &= p(\omega_1 \mid x_2) \cdot 1 + p(\omega_2 \mid x_2) \cdot 0 \\ &= \frac{p(x_2 \mid \omega_1) \cdot p(\omega_1)}{p(x_2)} \\ &= \frac{0.2 \cdot 0.9}{0.25} = 0.72 \end{aligned}$$

The Bayesian Doctor - Cntd.

- To summarize:

$$R(\alpha_1 | x_1) = 0.4$$

$$R(\alpha_2 | x_1) = 0.96$$

$$R(\alpha_1 | x_2) = 2.8$$

$$R(\alpha_2 | x_2) = 0.72$$

- Whenever we encounter an observation x , we can minimize the expected loss by minimizing the conditional risk.
- Makes sense: Doctor chooses hot tea if blood test is negative, and antibiotics otherwise.

Optimal Bayes Decision Strategies

- A **strategy** or **decision function** $\alpha(x)$ is a mapping from observations to actions.
- The **total risk** of a decision function is given by

$$E_{p(x)}[R(\alpha(x) | x)] = \sum_x p(x) \cdot R(\alpha(x) | x)$$

- A decision function is **optimal** if it minimizes the total risk. This optimal total risk is called **Bayes risk**.
- In the Bayesian doctor example:
 - Total risk if doctor always gives antibiotics(a_2): **0.9**
 - Bayes risk: **0.48** **How have we got it?**

Normal Random Variable and its discriminant functions

Outline

- Normal Random Variable
 - Properties
 - Discriminant functions

Why Normal Random Variables?

- Analytically tractable
- Works well when observation comes from a corrupted single prototype (μ)

The Univariate Normal Density

- x is a scalar (has dimension 1)

$$p(x) = \frac{1}{\sqrt{2\pi} \sigma} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right],$$

Where:

μ = mean (or expected value) of x

σ^2 = variance

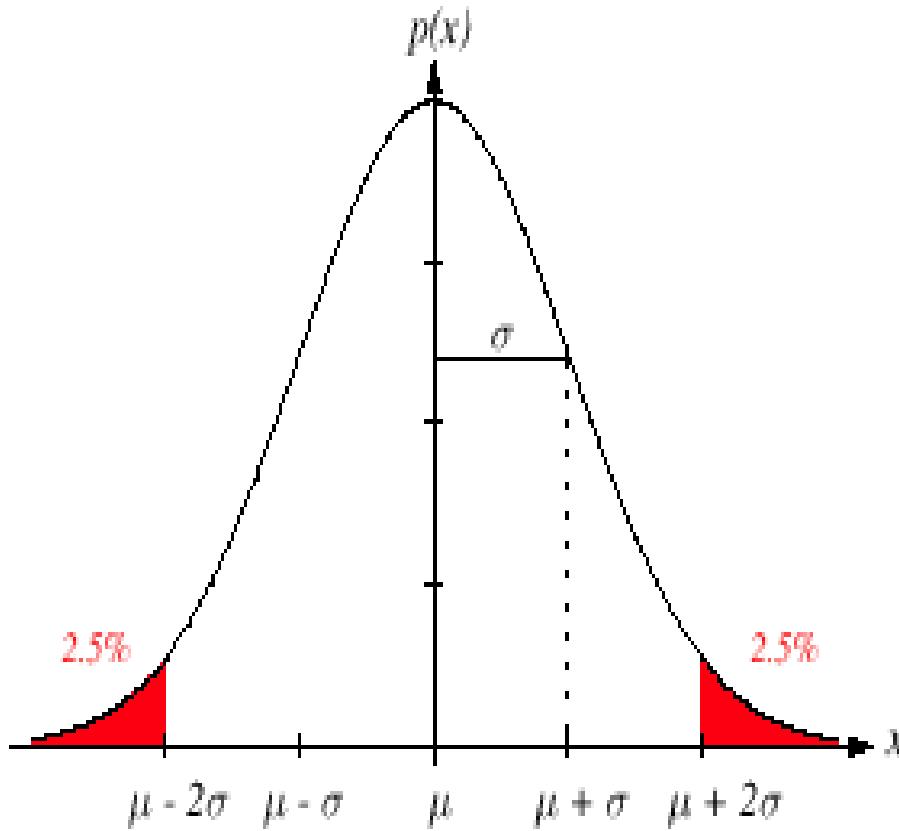


FIGURE 2.7. A univariate normal distribution has roughly 95% of its area in the range $|x - \mu| \leq 2\sigma$, as shown. The peak of the distribution has value $p(\mu) = 1/\sqrt{2\pi}\sigma$. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Several Features

- What if we have several features x_1, x_2, \dots, x_d
 - each normally distributed
 - may have different means
 - may have different variances
 - may be dependent or independent of each other
- How do we model their joint distribution?

The Multivariate Normal Density

- Multivariate normal density in d dimensions is:

$$p(x) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left[-\frac{1}{2} (x - \mu)^t \underset{\text{determinant of } \Sigma}{\Sigma^{-1}} (x - \mu) \right]$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_d^2 \end{bmatrix}$$

covariance of x_1 and x_d

$$x = [x_1, x_2, \dots, x_d]^t$$

$$\mu = [\mu_1, \mu_2, \dots, \mu_d]^t$$

- Each x_i is $N(\mu_i, \sigma_i^2)$

More on Σ

- $\Sigma = \begin{bmatrix} \sigma_1^2 & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_d^2 \end{bmatrix}$ plays role similar to the role that σ^2 plays in one dimension
- From Σ we can find out
 1. The individual variances of features x_1, x_2, \dots, x_d
 2. If features x_i and x_j are
 - independent $\sigma_{ij}=0$
 - have positive correlation $\sigma_{ij}>0$
 - have negative correlation $\sigma_{ij}<0$

The Multivariate Normal Density

- If Σ is diagonal $\begin{bmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{bmatrix}$ then the features x_i, \dots, x_j are independent, and

$$p(x) = \prod_{i=1}^d \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left[-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right]$$

The Multivariate Normal Density

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left[-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right]$$

$$p(\mathbf{x}) = c \cdot \exp \left[-\frac{1}{2} [\mathbf{x}_1 - \mu_1 \quad \mathbf{x}_2 - \mu_2 \quad \mathbf{x}_3 - \mu_3] \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{12} & \sigma_2^2 & \sigma_{23} \\ \sigma_{13} & \sigma_{23} & \sigma_3^2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{x}_1 - \mu_1 \\ \mathbf{x}_2 - \mu_2 \\ \mathbf{x}_3 - \mu_3 \end{bmatrix} \right]$$

normalizing

constant scalar s (single number), the closer s to 0 the larger is $p(x)$

- Thus $P(\mathbf{x})$ is larger for smaller $(\mathbf{x} - \boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$

$$(\mathbf{x} - \boldsymbol{\mu})^t \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$$

- $\boldsymbol{\Sigma}$ is positive semi definite ($\mathbf{x}^t \boldsymbol{\Sigma} \mathbf{x} \geq 0$)
- If $\mathbf{x}^t \boldsymbol{\Sigma} \mathbf{x} = 0$ for nonzero \mathbf{x} then $\det(\boldsymbol{\Sigma})=0$. This case is not interesting, $p(\mathbf{x})$ is not defined
 1. one feature vector is a constant (has zero variance)
 2. or two components are multiples of each other
- so we will assume $\boldsymbol{\Sigma}$ is positive definite ($\mathbf{x}^t \boldsymbol{\Sigma} \mathbf{x} > 0$)
- If $\boldsymbol{\Sigma}$ is positive definite then so is $\boldsymbol{\Sigma}^{-1}$

Eigenvalues/eigenvectors (from Wiki)

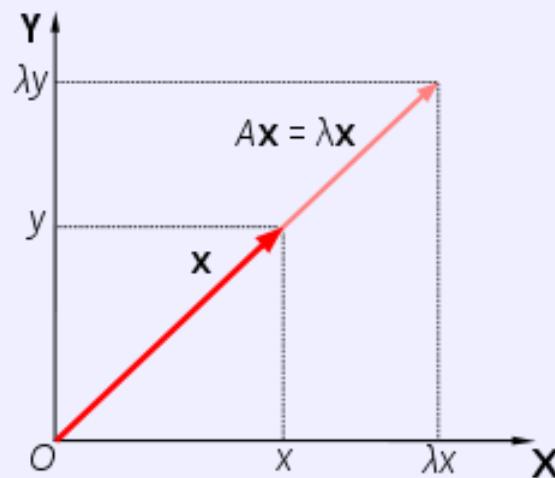
- Given a linear transformation A , a non-zero vector x is defined to be an eigenvector of the transformation if it satisfies the eigenvalue equation

$$Ax = \lambda x \text{ for some scalar } \lambda.$$

where λ is called an **eigenvalue** of A , corresponding to the **eigenvector** x .

Eigenvalues/eigenvectors (from Wiki)

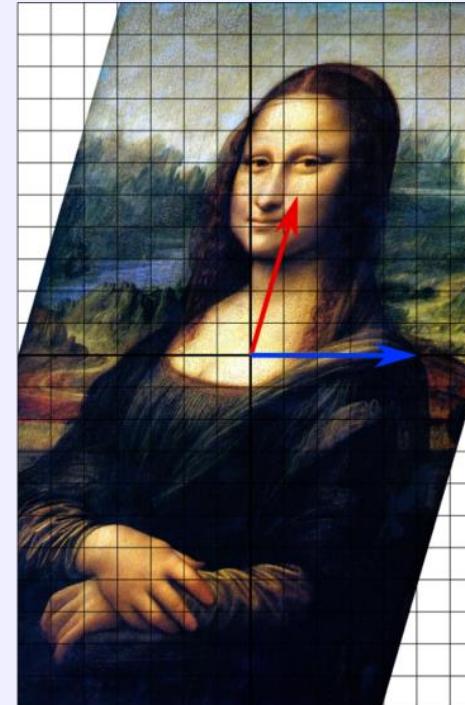
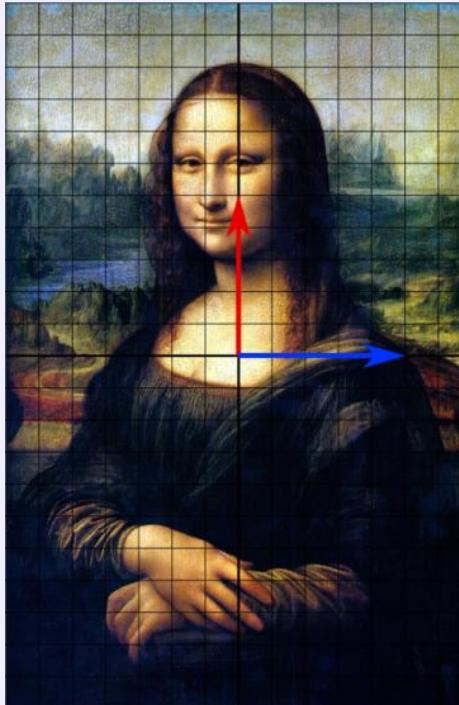
- Geometrically, it means that under the transformation A, eigenvectors only change in magnitude and sign—the direction of Ax is the same as that of x . The eigenvalue λ is simply the amount of "stretch" or "shrink" to which a vector is subjected when transformed by A.



- For example, an eigenvalue of +2 means that the eigenvector is doubled in length and points in the same direction. An eigenvalue of +1 means that the eigenvector is unchanged, while an eigenvalue of -1 means that the eigenvector is reversed in sense.

Eigenvalues/eigenvectors (from Wiki)

- In this shear mapping the red arrow changes direction but the blue arrow does not.
- Therefore the blue arrow is an eigenvector, with eigenvalue 1 as its length is unchanged.



$$(x - \mu)^t \Sigma^{-1} (x - \mu)$$

- Positive definite matrix of size d by d has d distinct real eigenvalues and its d eigenvectors are orthogonal
- Thus if Φ is a matrix whose columns are normalized eigenvectors of Σ , then $\Phi^{-1} = \Phi^t$
- $\Sigma \Phi = \Phi \Lambda$ where Λ is a diagonal matrix with corresponding eigenvalues on the diagonal
- Thus $\Sigma = \Phi \Lambda \Phi^{-1}$ and $\Sigma^{-1} = \Phi \Lambda^{-1} \Phi^{-1}$
- Thus if $\Lambda^{1/2}$ denotes matrix s.t. $\Lambda^{-1/2} \Lambda^{1/2} = \Lambda^{-1}$

$$\Sigma^{-1} = \left(\Phi \Lambda^{1/2} \right) \left(\Phi \Lambda^{1/2} \right)^t = M M^t$$

$$(x - \mu)^t \Sigma^{-1} (x - \mu)$$

- Thus

$$\begin{aligned}(x - \mu)^t \Sigma^{-1} (x - \mu) &= (x - \mu)^t M M^t (x - \mu) = \\ &= (M^t (x - \mu))^t (M^t (x - \mu)) = |M^t (x - \mu)|^2\end{aligned}$$

- Thus $(x - \mu)^t \Sigma^{-1} (x - \mu) = |M^t (x - \mu)|^2$

where $M^t = \Lambda^{-\frac{1}{2}} \Phi^{-1}$

*scaling rotation
matrix matrix*

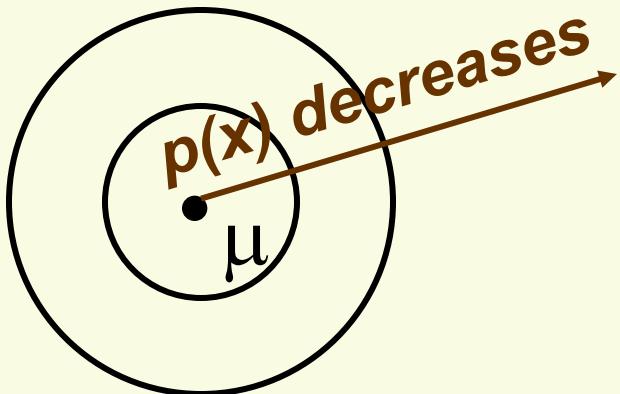
- Points x which satisfy $|M^t (x - \mu)|^2 = \text{const}$ lie on an ellipse

$$(x - \mu)^t \Sigma^{-1} (x - \mu)$$

$$(x - \mu)^t (x - \mu)$$

usual (Euclidian)

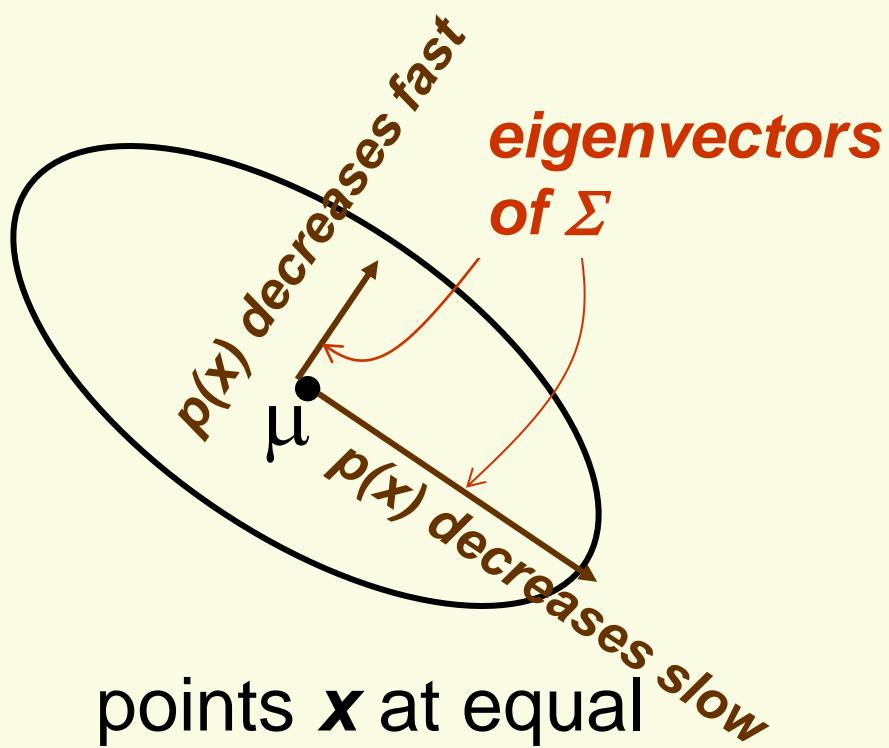
distance between x and μ



points x at equal
Euclidian
distance from μ
lie on a circle

$$(x - \mu)^t \Sigma^{-1} (x - \mu)$$

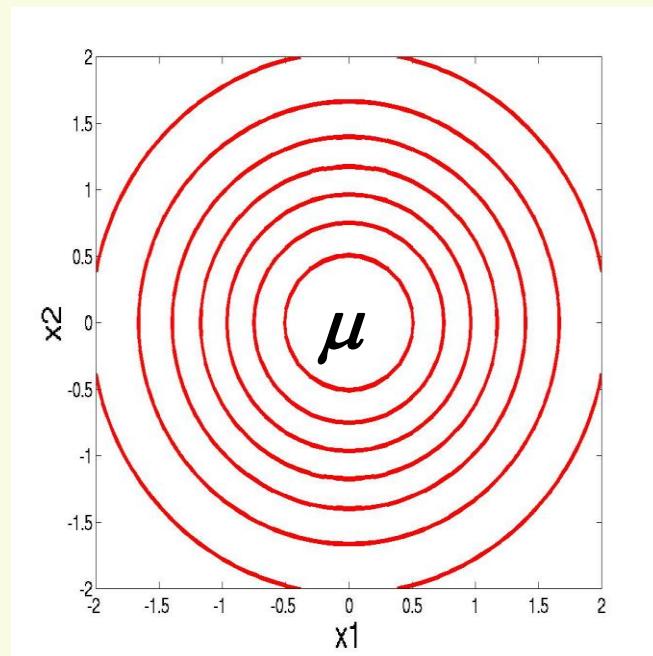
*Mahalanobis distance
between x and μ*



points x at equal
Mahalanobis distance from
 μ lie on an ellipse: Σ
stretches circles to ellipses

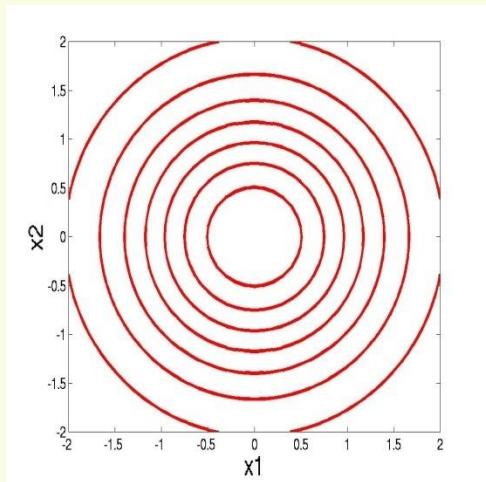
2-d Multivariate Normal Density

- Level curves graph
 - $p(\mathbf{x})$ is constant along each contour
 - topological map of 3-d surface

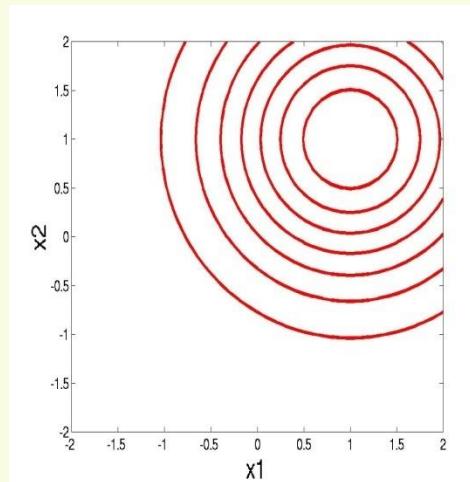


- x_1 and x_2 are independent
- σ_1^2 and σ_2^2 are equal

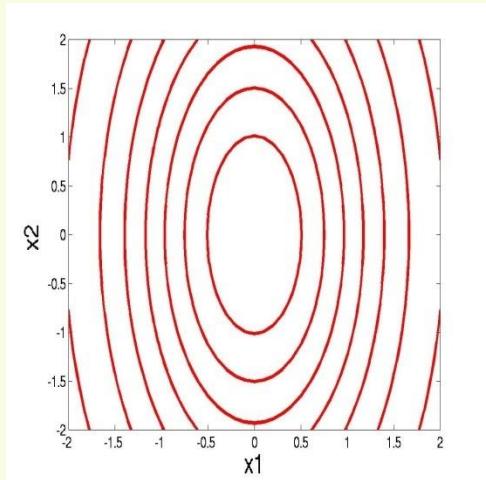
2-d Multivariate Normal Density



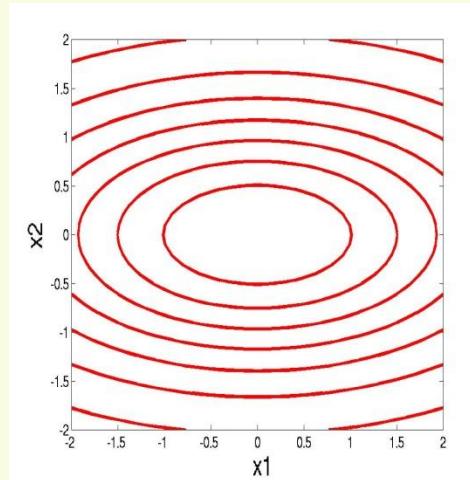
$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\mu = [0,0]$$



$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\mu = [1,1]$$

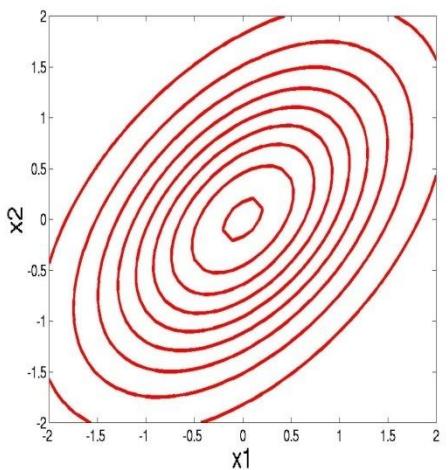


$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}$$
$$\mu = [0,0]$$

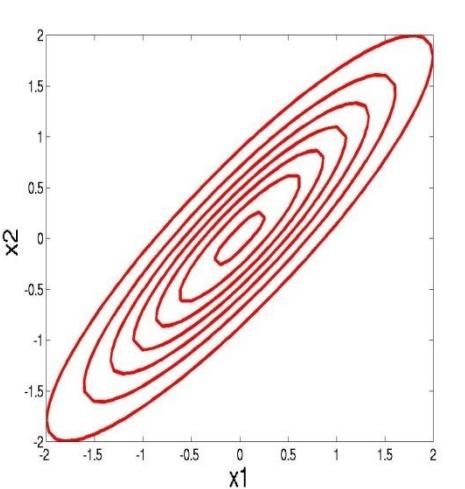


$$\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$
$$\mu = [0,0]$$

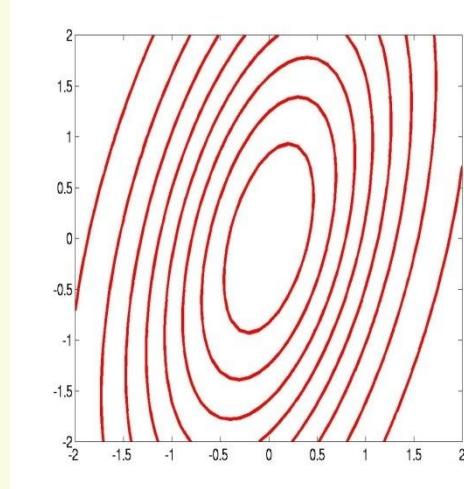
2-d Multivariate Normal Density $\mu = [0,0]$



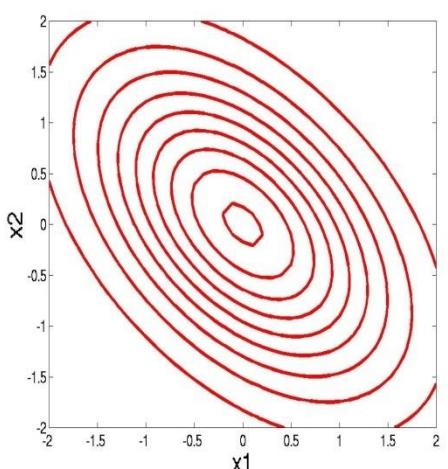
$$\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$



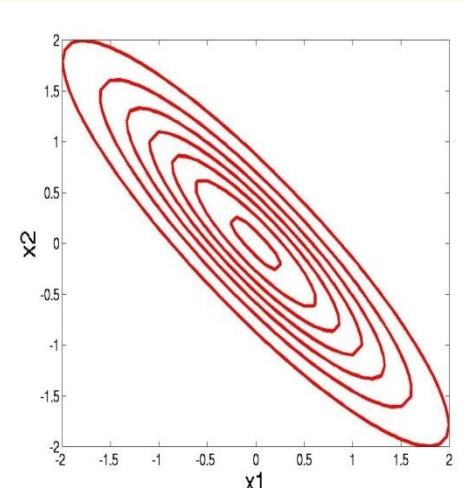
$$\Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$



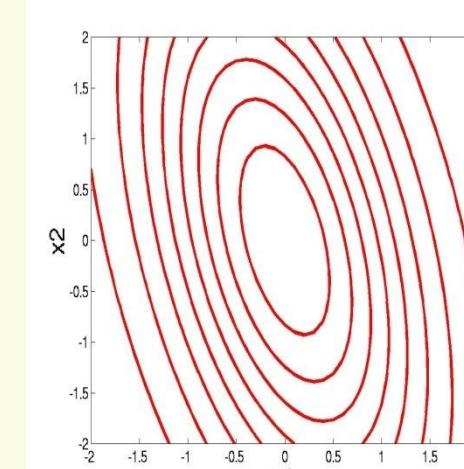
$$\Sigma = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 4 \end{bmatrix}$$



$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$



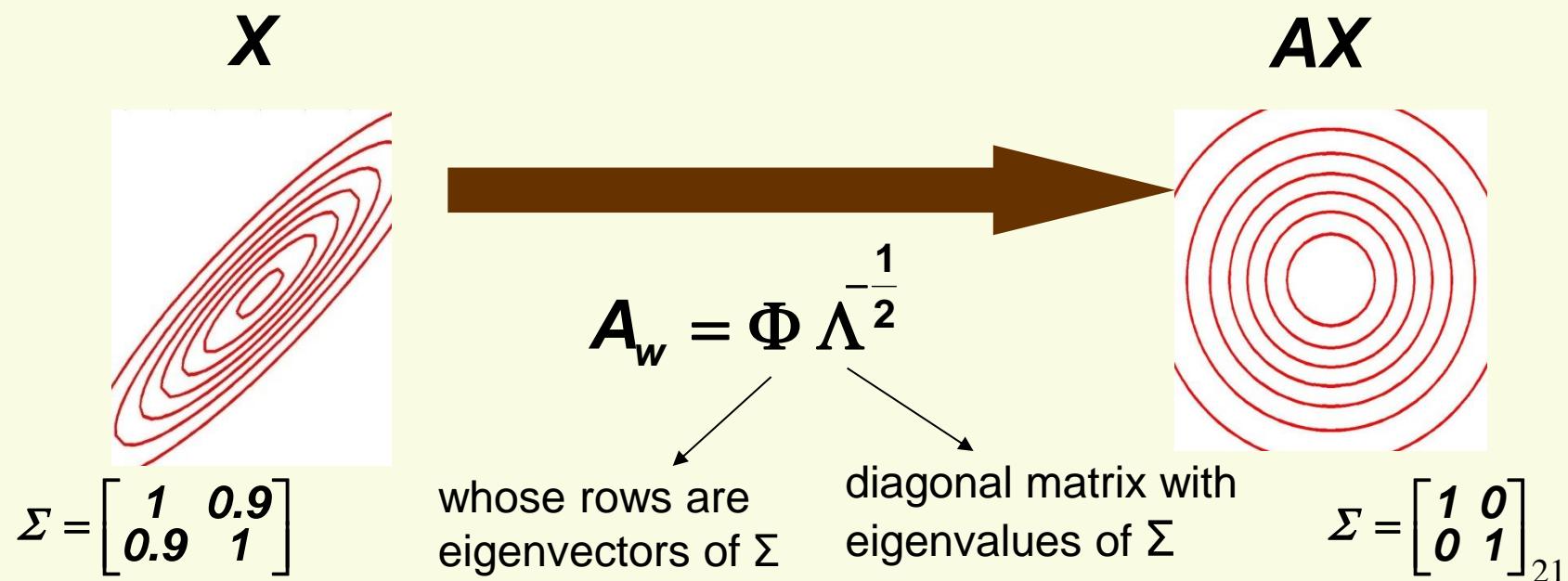
$$\Sigma = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}$$



$$\Sigma = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 4 \end{bmatrix}$$

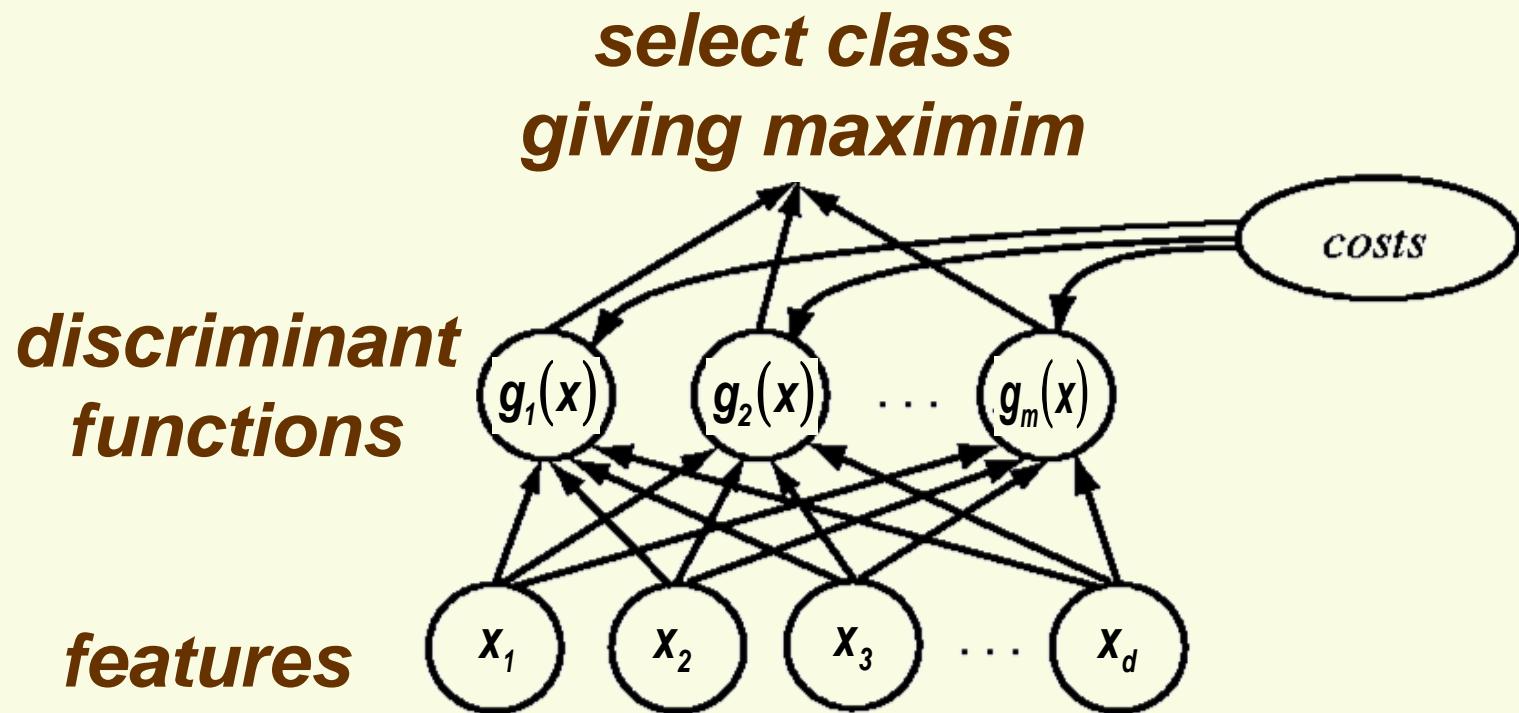
The Multivariate Normal Density

- If X has density $N(\mu, \Sigma)$ then AX has density $N(A^t\mu, A^t\Sigma A)$
 - Thus X can be transformed into a spherical normal variable (covariance of spherical density is the identity matrix I) with whitening transform



Discriminant Functions

- Classifier can be viewed as network which computes m discriminant functions and selects category corresponding to the largest discriminant



- $g_i(x)$ can be replaced with any monotonically increasing function of g , the results will be unchanged

Discriminant Functions

- The minimum error-rate classification is achieved by the discriminant function

$$g_i(x) = P(c_i | x) = P(x|c_i)P(c_i)/P(x)$$

- Since the observation x is independent of the class, the equivalent discriminant function is

$$g_i(x) = P(x|c_i)P(c_i)$$

- For normal density, convenient to take logarithms. Since logarithm is a monotonically increasing function, the equivalent discriminant function is

$$g_i(x) = \ln P(x|c_i) + \ln P(c_i)$$

Discriminant Functions for the Normal Density

- Suppose for class c_i its class conditional density $p(x|c_i)$ is $N(\mu_i, \Sigma_i)$

$$p(x|c_i) = \frac{1}{(2\pi)^{d/2} |\Sigma_i|^{1/2}} \exp \left[-\frac{1}{2} (x - \mu_i)^t \Sigma_i^{-1} (x - \mu_i) \right]$$

- Discriminant function $g_i(x) = \ln P(x|c_i) + \ln P(c_i)$
- Plug in $p(x|c_i)$ and $P(c_i)$ get

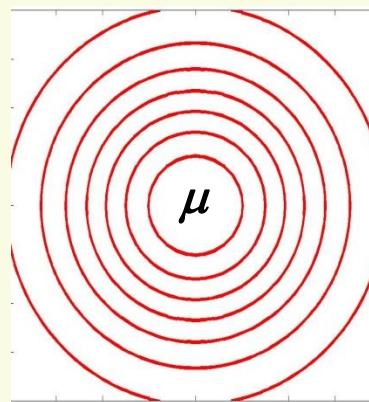
$$g_i(x) = -\frac{1}{2} (x - \mu_i)^t \Sigma_i^{-1} (x - \mu_i) - \frac{d}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i)$$

constant for all i

$$g_i(x) = -\frac{1}{2} (x - \mu_i)^t \Sigma_i^{-1} (x - \mu_i) - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i)$$

Case $\Sigma_i = \sigma^2 I$

- That is $\Sigma_i = \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix} = \sigma^2 \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- In this case, features x_1, x_2, \dots, x_d are independent with different means and equal variances σ^2



Case $\Sigma_i = \sigma^2 I$

- Discriminant function

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^t \sum^{-1}(x - \mu_i) - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i)$$

- $\text{Det}(\Sigma_i) = \sigma^{2d}$ and $\Sigma_i^{-1} = (1/\sigma^2)I = \begin{bmatrix} \frac{1}{\sigma^2} & 0 & 0 \\ 0 & \frac{1}{\sigma^2} & 0 \\ 0 & 0 & \frac{1}{\sigma^2} \end{bmatrix}$

- Can simplify discriminant function

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^t \frac{I}{\sigma^2} (x - \mu_i) - \frac{1}{2} \ln(\sigma^{2d}) + \ln P(c_i)$$

constant for all i

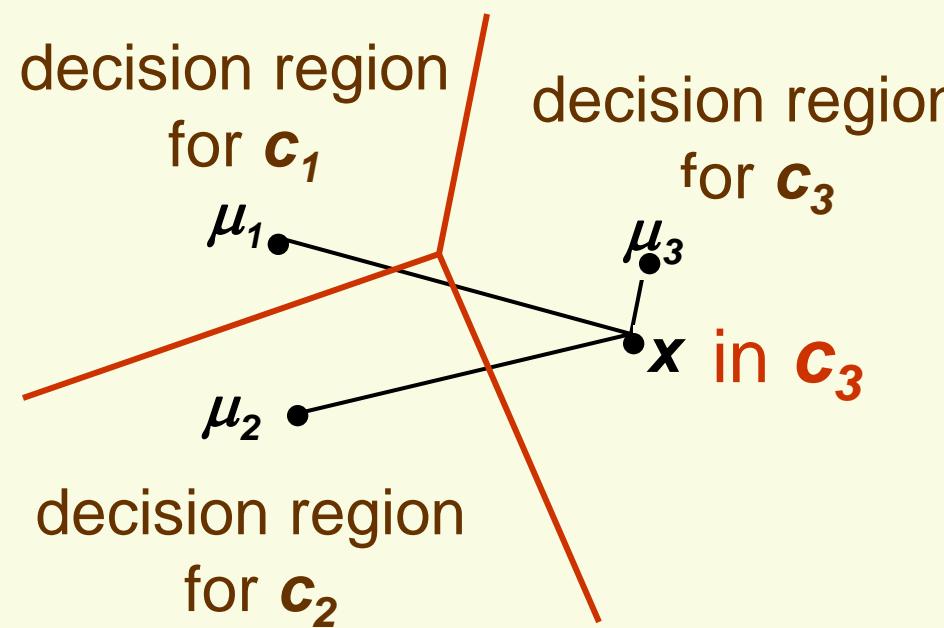
$$g_i(x) = -\frac{1}{2\sigma^2} (x - \mu_i)^t (x - \mu_i) + \ln P(c_i) =$$

$$= -\frac{1}{2\sigma^2} |x - \mu_i|^2 + \ln P(c_i)$$

Case $\Sigma_i = \sigma^2 I$ Geometric Interpretation

If $\ln P(c_i) = \ln P(c_j)$, then

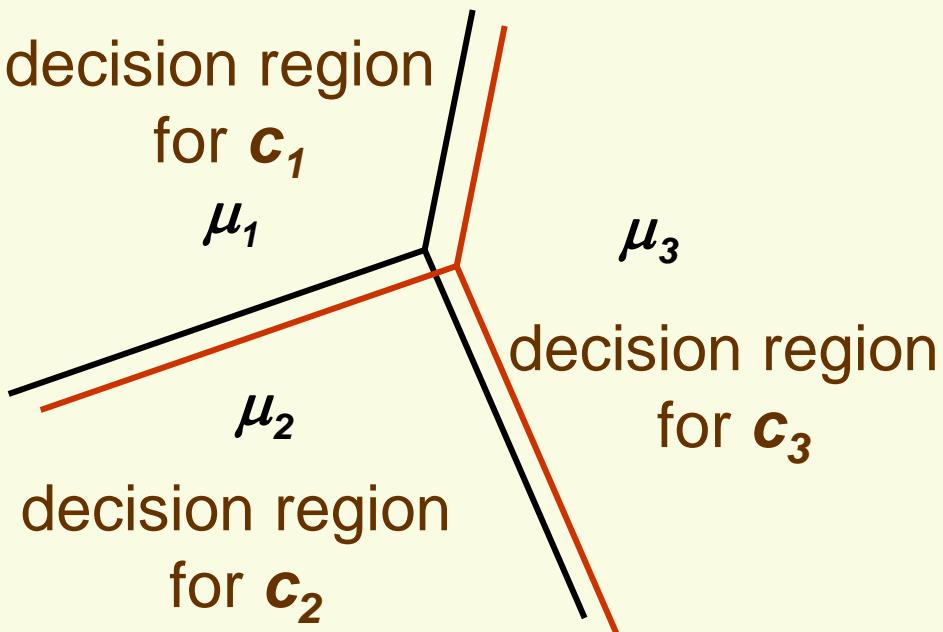
$$g_i(x) = -|x - \mu_i|^2$$



voronoi diagram: points in each cell are closer to the mean in that cell than to any other mean

If $\ln P(c_i) \neq \ln P(c_j)$, then

$$g_i(x) = -\frac{1}{2\sigma^2} |x - \mu_i|^2 + \ln P(c_i)$$



Case $\Sigma_i = \sigma^2 I$

$$\begin{aligned}g_i(x) &= -\frac{1}{2\sigma^2}(x - \mu_i)^t(x - \mu_i) + \ln P(c_i) = \\&= -\frac{1}{2\sigma^2}(\cancel{x^t x} - \mu_i^t x - x^t \mu_i + \mu_i^t \mu_i) + \ln P(c_i) \\&\quad \text{constant} \\&\quad \text{for all classes}\end{aligned}$$

$$g_i(x) = -\frac{1}{2\sigma^2}(-2\mu_i^t x + \mu_i^t \mu_i) + \ln P(c_i) = \frac{\mu_i^t}{\sigma^2} x + \left(-\frac{\mu_i^t \mu_i}{2\sigma^2} + \ln P(c_i)\right)$$
$$g_i(x) = \mathbf{w}_i^t \mathbf{x} + w_{i0}$$

discriminant function is linear

Case $\Sigma_j = \sigma^2 I$

$$g_i(x) = \mathbf{w}_i^t \mathbf{x} + w_{i0}$$

constant in x

linear in x :

$$\mathbf{w}_i^t \mathbf{x} = \sum_{i=1}^d w_i x_i$$

- Thus discriminant function is linear,
- Therefore the decision boundaries $g_i(\mathbf{x}) = g_j(\mathbf{x})$ are linear
 - lines if \mathbf{x} has dimension 2
 - planes if \mathbf{x} has dimension 3
 - hyper-planes if \mathbf{x} has dimension larger than 3

Case $\Sigma_i = \sigma^2 I$: Example

- 3 classes, each 2-dimensional Gaussian with

$$\mu_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \mu_2 = \begin{bmatrix} 4 \\ 6 \end{bmatrix} \quad \mu_3 = \begin{bmatrix} -2 \\ 4 \end{bmatrix} \quad \Sigma_1 = \Sigma_2 = \Sigma_3 = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$$

- Priors $P(c_1) = P(c_2) = \frac{1}{4}$ and $P(c_3) = \frac{1}{2}$

- Discriminant function is $g_i(x) = \frac{\mu_i^t}{\sigma^2} x + \left(-\frac{\mu_i^t \mu_i}{2\sigma^2} + \ln P(c_i) \right)$

- Plug in parameters for each class

$$g_1(x) = \frac{[1 \ 2]}{3} x + \left(-\frac{5}{6} - 1.38 \right) \quad g_2(x) = \frac{[4 \ 6]}{3} x + \left(-\frac{52}{6} - 1.38 \right)$$

$$g_3(x) = \frac{[-2 \ 4]}{3} x + \left(-\frac{20}{6} - 0.69 \right)$$

Case $\Sigma_i = \sigma^2 I$: Example

- Need to find out when $\mathbf{g}_i(\mathbf{x}) < \mathbf{g}_j(\mathbf{x})$ for $i,j=1,2,3$
- Can be done by solving $\mathbf{g}_i(\mathbf{x}) = \mathbf{g}_j(\mathbf{x})$ for $i,j=1,2,3$
- Let's take $\mathbf{g}_1(\mathbf{x}) = \mathbf{g}_2(\mathbf{x})$ first

$$\frac{\begin{bmatrix} 1 & 2 \end{bmatrix}}{3} \mathbf{x} + \left(-\frac{5}{6} - 1.38 \right) = \frac{\begin{bmatrix} 4 & 6 \end{bmatrix}}{3} \mathbf{x} + \left(-\frac{52}{6} - 1.38 \right)$$

- Simplifying, $\frac{[-3 - 4]}{3} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = -\frac{47}{6}$

$$-\mathbf{x}_1 - \frac{4}{3} \mathbf{x}_2 = -\frac{47}{6}$$

line equation

Case $\Sigma_i = \sigma^2 I$: Example

- Next solve $\mathbf{g}_2(\mathbf{x}) = \mathbf{g}_3(\mathbf{x})$

$$2x_1 + \frac{2}{3}x_2 = 6.02$$

- Almost finally solve $\mathbf{g}_1(\mathbf{x}) = \mathbf{g}_3(\mathbf{x})$

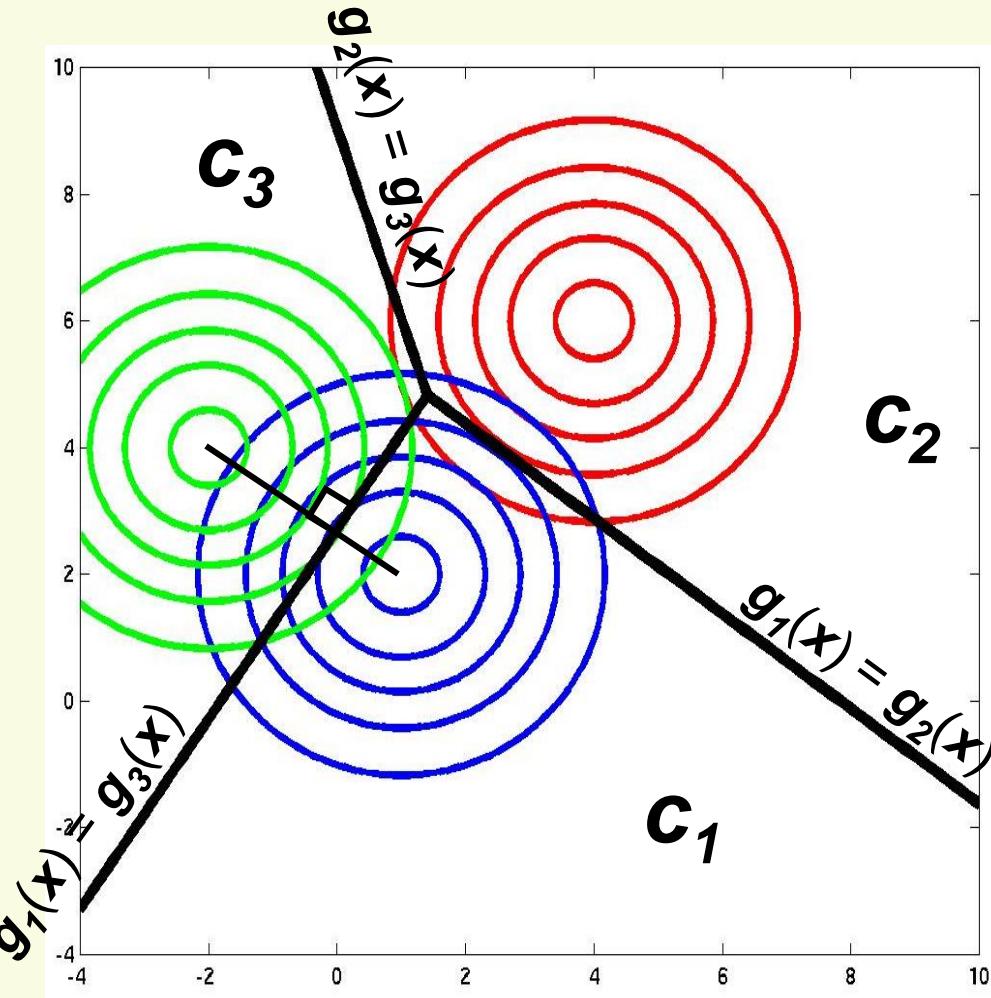
$$x_1 - \frac{2}{3}x_2 = -1.81$$

- And finally solve $\mathbf{g}_1(\mathbf{x}) = \mathbf{g}_2(\mathbf{x}) = \mathbf{g}_3(\mathbf{x})$

$$x_1 = 1.4 \quad \text{and} \quad x_2 = 4.82$$

Case $\Sigma_i = \sigma^2 I$: Example

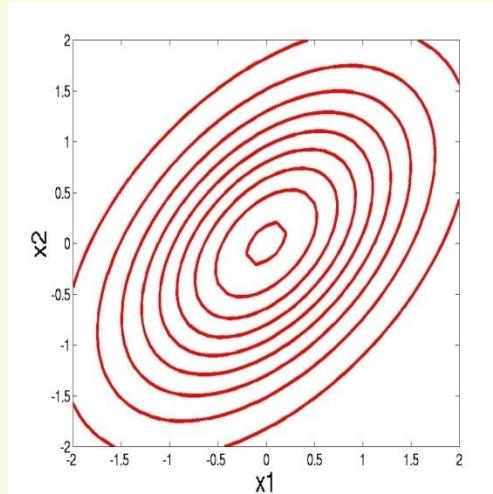
- Priors $P(c_1) = P(c_2) = \frac{1}{4}$ and $P(c_3) = \frac{1}{2}$



*lines connecting
means
are perpendicular to
decision boundaries*

Case $\Sigma_i = \Sigma$

- Covariance matrices are equal but arbitrary
- In this case, features x_1, x_2, \dots, x_d are not necessarily independent



$$\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

Case $\Sigma_i = \Sigma$

- Discriminant function

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^t \sum^{-1}(x - \mu_i) - \frac{1}{2} \cancel{\ln |\Sigma_i|} + \ln P(c_i)$$

*constant
for all classes*

- Discriminant function becomes

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^t \sum^{-1}(x - \mu_i) + \ln P(c_i)$$

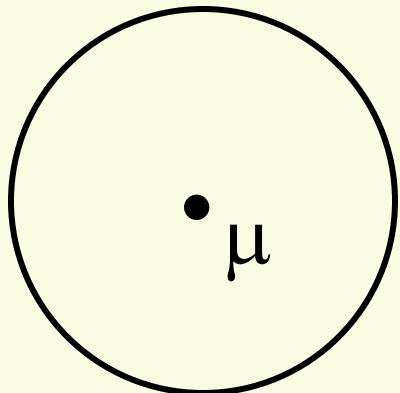
squared Mahalanobis Distance

- Mahalanobis Distance $\|x - y\|_{\Sigma^{-1}}^2 = (x - y)^t \sum^{-1}(x - y)$
- If $\Sigma = I$, Mahalanobis Distance becomes usual Euclidian distance

$$\|x - y\|_{I^{-1}}^2 = (x - y)^t (x - y)$$

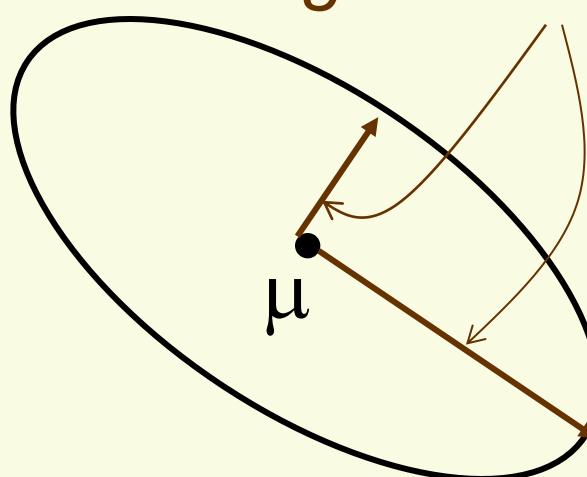
Eucledian vs. Mahalanobis Distances

$$|x - \mu|^2 = (x - \mu)^t(x - \mu)$$



points x at equal
Eucledian
distance from μ
lie on a circle

$$\|x - \mu\|_{\Sigma^{-1}}^2 = (x - \mu)^t \Sigma^{-1} (x - \mu)$$

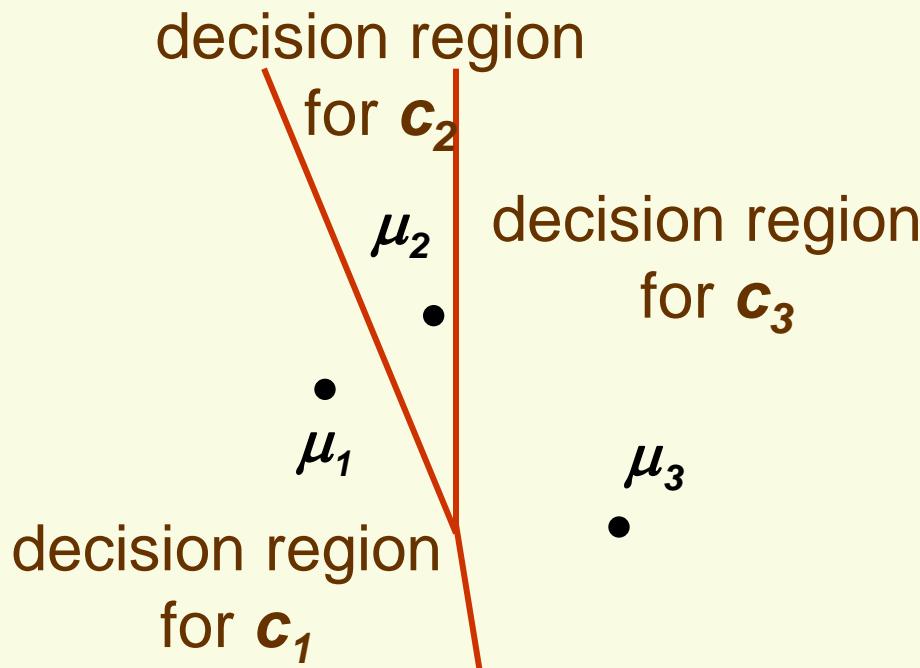


points x at equal
Mahalanobis distance from
 μ lie on an ellipse:
 Σ stretches circles to ellipses

Case $\Sigma_i = \Sigma$ Geometric Interpretation

If $\ln P(c_i) = \ln P(c_j)$, then

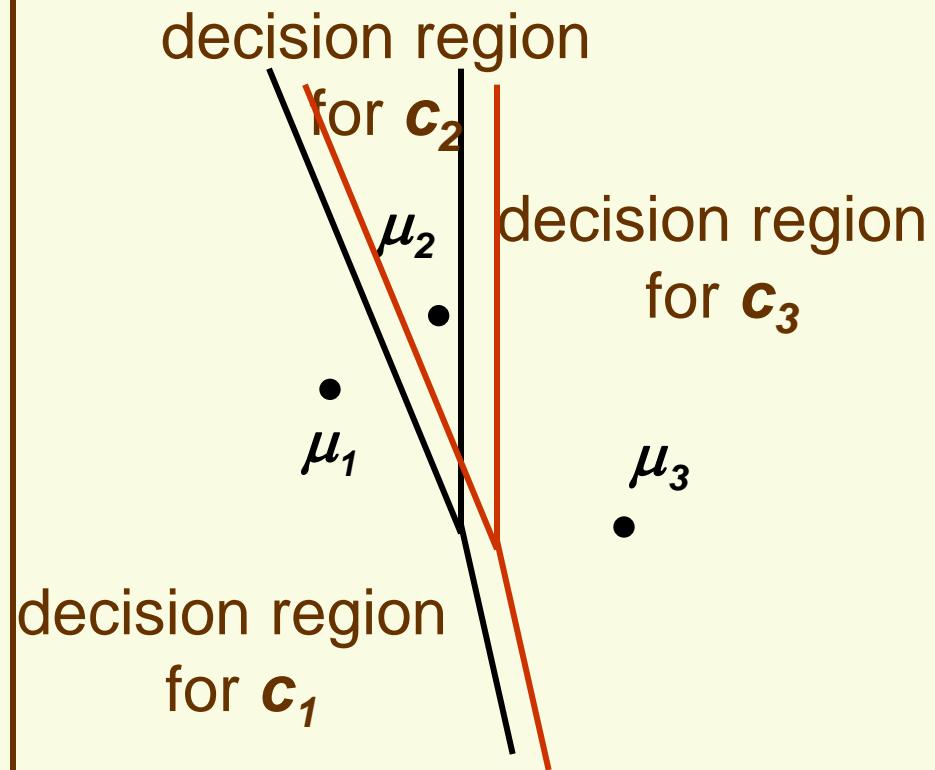
$$g_i(x) = -\|x - \mu_i\|_{\Sigma^{-1}}$$



points in each cell are closer to the mean in that cell than to any other mean under Mahalanobis distance

If $\ln P(c_i) \neq \ln P(c_j)$, then

$$g_i(x) = -\frac{1}{2}\|x - \mu_i\|_{\Sigma^{-1}} + \ln P(c_i)$$



Case $\Sigma_i = \Sigma$

- Can simplify discriminant function:

$$\begin{aligned} g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}_i) + \ln P(c_i) = \\ &= -\frac{1}{2} \left(\mathbf{x}^t \boldsymbol{\Sigma}^{-1} \mathbf{x} - \boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \mathbf{x} - \mathbf{x}^t \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \right) + \ln P(c_i) = \\ &= -\frac{1}{2} \left(\cancel{\mathbf{x}^t \boldsymbol{\Sigma}^{-1} \mathbf{x}} - 2\boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \right) + \ln P(c_i) = \\ &\quad \text{constant for all classes} \\ &= -\frac{1}{2} \left(-2\boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \mathbf{x} + \boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \right) + \ln P(c_i) \\ &= \boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \mathbf{x} + \left(\ln P(c_i) - \frac{1}{2} \boldsymbol{\mu}_i^t \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \right) = \mathbf{w}_i^t \mathbf{x} + w_{i0} \end{aligned}$$

- Thus in this case discriminant is also linear

Case $\Sigma_i = \Sigma$: Example

- 3 classes, each 2-dimensional Gaussian with

$$\mu_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \mu_2 = \begin{bmatrix} -1 \\ 5 \end{bmatrix} \quad \mu_3 = \begin{bmatrix} -2 \\ 4 \end{bmatrix} \quad \Sigma_1 = \Sigma_2 = \Sigma_3 = \begin{bmatrix} 1 & -1.5 \\ -1.5 & 4 \end{bmatrix}$$

$$P(c_1) = P(c_2) = \frac{1}{4} \qquad \qquad P(c_3) = \frac{1}{2}$$

- Again can be done by solving $\mathbf{g}_i(\mathbf{x}) = \mathbf{g}_j(\mathbf{x})$ for $i, j = 1, 2, 3$

Case $\Sigma_j = \Sigma$: Example

- Let's solve in general first

$$g_j(x) = g_i(x)$$

$$\mu_j^t \Sigma^{-1} x + \left(\ln P(c_j) - \frac{1}{2} \mu_j^t \Sigma^{-1} \mu_j \right) = \mu_i^t \Sigma^{-1} x + \left(\ln P(c_i) - \frac{1}{2} \mu_i^t \Sigma^{-1} \mu_i \right)$$

- Let's regroup the terms

$$(\mu_j^t \Sigma^{-1} - \mu_i^t \Sigma^{-1})x = -\left(\ln P(c_j) - \frac{1}{2} \mu_j^t \Sigma^{-1} \mu_j \right) + \left(\ln P(c_i) - \frac{1}{2} \mu_i^t \Sigma^{-1} \mu_i \right)$$

- We get the line where $g_j(x) = g_i(x)$

$$(\mu_j^t - \mu_i^t) \Sigma^{-1} x = \left(\ln \frac{P(c_i)}{P(c_j)} + \frac{1}{2} \mu_j^t \Sigma^{-1} \mu_j - \frac{1}{2} \mu_i^t \Sigma^{-1} \mu_i \right)$$

row vector *scalar*

Case $\Sigma_i = \Sigma$: Example

$$(\mu_j^t - \mu_i^t) \Sigma^{-1} x = \left(\ln \frac{P(c_i)}{P(c_j)} + \frac{1}{2} \mu_j^t \Sigma^{-1} \mu_j - \frac{1}{2} \mu_i^t \Sigma^{-1} \mu_i \right)$$

- Now substitute for i,j=1,2

$$\begin{bmatrix} -2 & 0 \end{bmatrix} x = 0$$

$$x_1 = 0$$

- Now substitute for i,j=2,3

$$\begin{bmatrix} -3.14 & -1.4 \end{bmatrix} x = -2.41$$

$$3.14x_1 + 1.4x_2 = 2.41$$

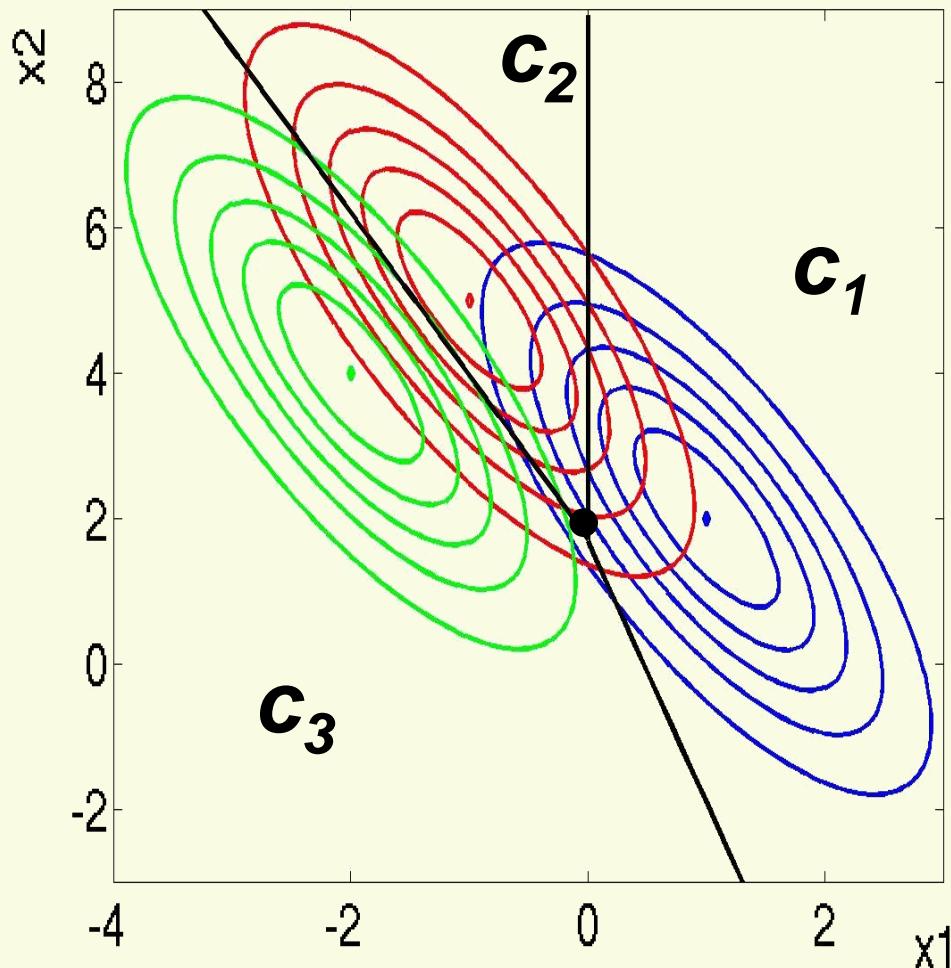
- Now substitute for i,j=1,3

$$\begin{bmatrix} -5.14 & -1.43 \end{bmatrix} x = -2.41$$

$$5.14x_1 + 1.43x_2 = 2.41$$

Case $\Sigma_i = \Sigma$: Example

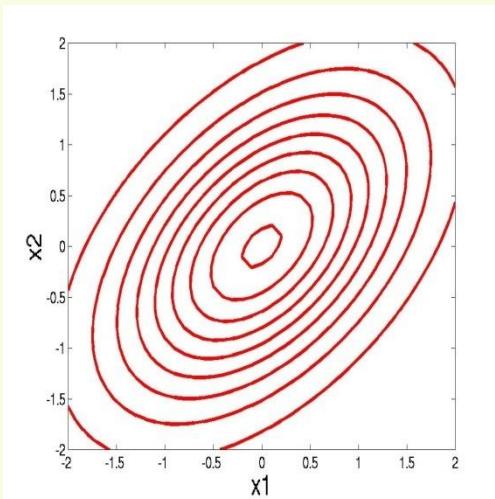
- Priors $P(c_1) = P(c_2) = \frac{1}{4}$ and $P(c_3) = \frac{1}{2}$



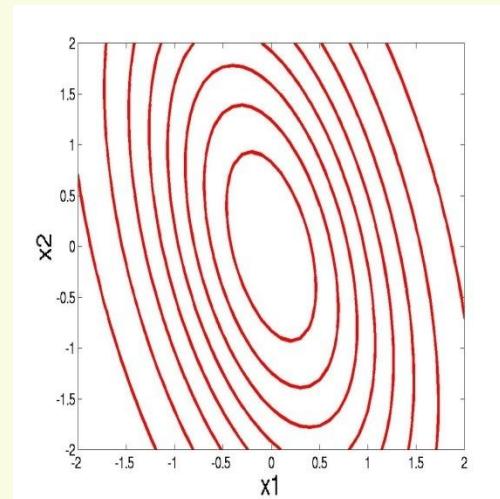
*lines connecting means are **not** in general perpendicular to decision boundaries*

General Case Σ_i are arbitrary

- Covariance matrices for each class are arbitrary
- In this case, features x_1, x_2, \dots, x_d are not necessarily independent



$$\Sigma_i = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$



$$\Sigma_j = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 4 \end{bmatrix}$$

General Case Σ_i are arbitrary

- From previous discussion,

$$g_i(x) = -\frac{1}{2}(x - \mu_i)^t \Sigma_i^{-1} (x - \mu_i) - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i)$$

- This can't be simplified, but we can rearrange it:

$$g_i(x) = -\frac{1}{2}(x^t \Sigma_i^{-1} x - 2\mu_i^t \Sigma_i^{-1} x + \mu_i^t \Sigma_i^{-1} \mu_i) - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i)$$

$$g_i(x) = x^t \left(-\frac{1}{2} \Sigma_i^{-1} \right) x + \mu_i^t \Sigma_i^{-1} x + \left(-\frac{1}{2} \mu_i^t \Sigma_i^{-1} \mu_i - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i) \right)$$

$$g_i(x) = x^t W x + w^t x + w_{i0}$$

General Case Σ_i are arbitrary

$$g_i(x) = \mathbf{x}^t \mathbf{W} \mathbf{x} + \mathbf{w}^t \mathbf{x} + w_{i0}$$

linear in x

constant in x

quadratic in x since

$$\mathbf{x}^t \mathbf{W} \mathbf{x} = \sum_{j=1}^d \sum_{i=1}^d w_{ij} x_i x_j = \sum_{i,j=1}^d w_{ij} x_i x_j$$

- Thus the discriminant function is quadratic
- Therefore the decision boundaries are quadratic (ellipses and paraboloids)

General Case Σ_i are arbitrary: Example

- 3 classes, each 2-dimensional Gaussian with

$$\mu_1 = \begin{bmatrix} -1 \\ 3 \end{bmatrix} \quad \mu_2 = \begin{bmatrix} 0 \\ 6 \end{bmatrix} \quad \mu_3 = \begin{bmatrix} -2 \\ 4 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 2 \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 2 & -2 \\ -2 & 7 \end{bmatrix} \quad \Sigma_3 = \begin{bmatrix} 1 & 1.5 \\ 1.5 & 3 \end{bmatrix}$$

- Priors: $P(c_1) = P(c_2) = \frac{1}{4}$ and $P(c_3) = \frac{1}{2}$
- Again can be done by solving $g_i(\mathbf{x}) = g_j(\mathbf{x})$ for $i, j = 1, 2, 3$

$$g_i(\mathbf{x}) = \mathbf{x}^t \left(-\frac{1}{2} \Sigma_i^{-1} \right) \mathbf{x} + \mu_i^t \Sigma_i^{-1} \mathbf{x} + \left(-\frac{1}{2} \mu_i^t \Sigma_i^{-1} \mu_i - \frac{1}{2} \ln |\Sigma_i| + \ln P(c_i) \right)$$

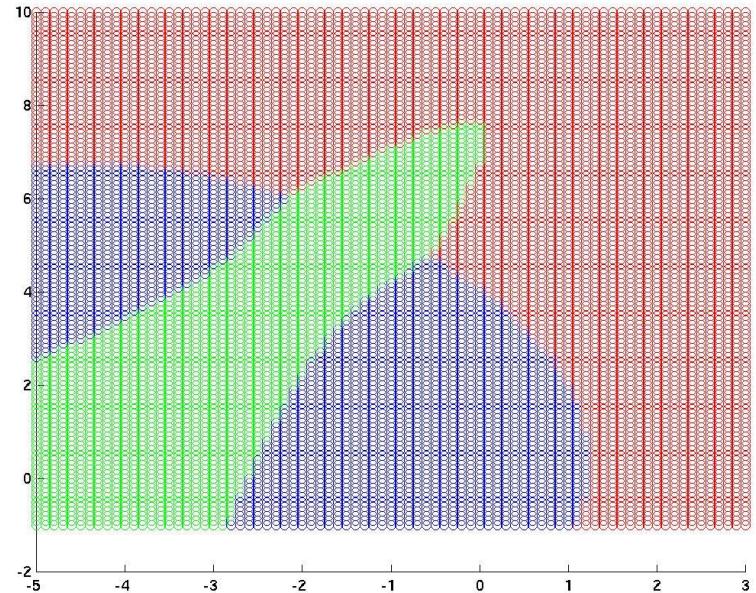
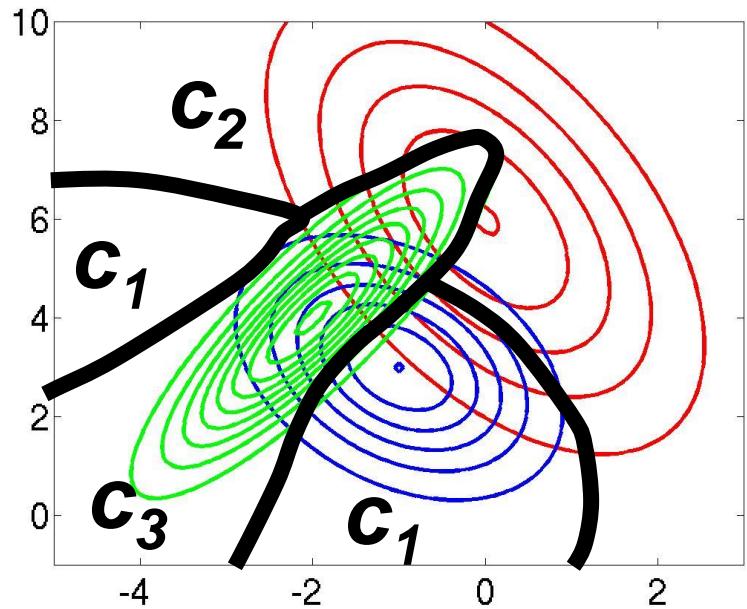
- Need to solve a bunch of quadratic inequalities of 2 variables

General Case Σ_i are arbitrary: Example

$$\mu_1 = \begin{bmatrix} -1 \\ 3 \end{bmatrix} \quad \mu_2 = \begin{bmatrix} 0 \\ 6 \end{bmatrix} \quad \mu_3 = \begin{bmatrix} -2 \\ 4 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 2 \end{bmatrix} \quad \Sigma_2 = \begin{bmatrix} 2 & -2 \\ -2 & 7 \end{bmatrix} \quad \Sigma_3 = \begin{bmatrix} 1.5 & 1.5 \\ 1.5 & 3 \end{bmatrix}$$

$$P(c_1) = P(c_2) = \frac{1}{4} \quad P(c_3) = \frac{1}{2}$$



Important Points

- The Bayes classifier when classes are normally distributed is in general quadratic
 - If covariance matrices are equal and proportional to identity matrix, the Bayes classifier is linear
 - If, in addition the priors on classes are equal, the Bayes classifier is the minimum Euclidian distance classifier
 - If covariance matrices are equal, the Bayes classifier is linear
 - If, in addition the priors on classes are equal, the Bayes classifier is the minimum Mahalanobis distance classifier
- Popular classifiers (Euclidean and Mahalanobis distance) are optimal only if distribution of data is appropriate (normal)

Parametric Density Estimation:

Maximum Likelihood Estimation

Introduction

- Bayesian Decision Theory in previous lectures tells us how to design an optimal classifier if we knew:
 - $P(c_i)$ (priors)
 - $P(x | c_i)$ (class-conditional densities)
- Unfortunately, we rarely have this complete information!

Probability density methods

- Parametric methods – assume we know the shape of the distribution, but not the parameters. Two types of parameter estimation:
 - Maximum Likelihood Estimation
 - Bayesian Estimation
- Non parametric methods – the form of the density is entirely determined by the data without any model.

Independence Across Classes

- We have training data for each class

salmon



sea bass



salmon



salmon



sea bass



sea bass



- When estimating parameters for one class, will only use the data collected for that class
 - reasonable assumption that data from class c_i gives no information about distribution of class c_j

estimate parameters for distribution of salmon from



estimate parameters for distribution of bass from



Independence Across Classes

- For each class c_i , we have a proposed density $p_i(\mathbf{x} | c_i)$ with unknown parameters θ^i which we need to estimate
- Since we assumed independence of data across the classes, estimation is an identical procedure for all classes
- To simplify notation, we drop sub-indexes and say that we need to estimate parameters θ for density $p(\mathbf{x})$
 - the fact that we need to do so for each class on the training data that came from that class is implied

Maximum Likelihood Parameter Estimation

- Parameters θ are unknown but fixed (i.e. not random variables).
- Given the training data, choose the parameter value θ that makes the data most probable (i.e., maximizes the probability of obtaining the sample that has actually been observed)

Maximum Likelihood Parameter Estimation

- We have density $p(\mathbf{x})$ which is completely specified by parameters $\theta = [\theta_1, \dots, \theta_k]$
 - If $p(\mathbf{x})$ is $N(\mu, \sigma^2)$ then $\theta = [\mu, \sigma^2]$
- To highlight that $p(\mathbf{x})$ depends on parameters θ we will write $p(\mathbf{x}|\theta)$
 - Note overloaded notation, $p(\mathbf{x}|\theta)$ is **not** a conditional density
- Let $D = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ be the n independent training samples in our data
 - If $p(\mathbf{x})$ is $N(\mu, \sigma^2)$ then $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ are iid samples from $N(\mu, \sigma^2)$

Maximum Likelihood Parameter Estimation

- Consider the following function, which is called **likelihood of θ** with respect to the set of samples D

$$p(D|\theta) = \prod_{k=1}^{k=n} p(x_k|\theta) = F(\theta)$$

- Maximum likelihood estimate (abbreviated **MLE**) of θ is the value of θ that maximizes the likelihood function $p(D|\theta)$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}(p(D|\theta))$$

ML Parameter Estimation vs. ML Classifier

- Recall ML classifier

decide class c_i , which maximizes $p(x|c_i)$

*fixed
data*

- Compare with ML parameter estimation

choose θ that maximizes $p(D|\theta)$

*fixed
data*

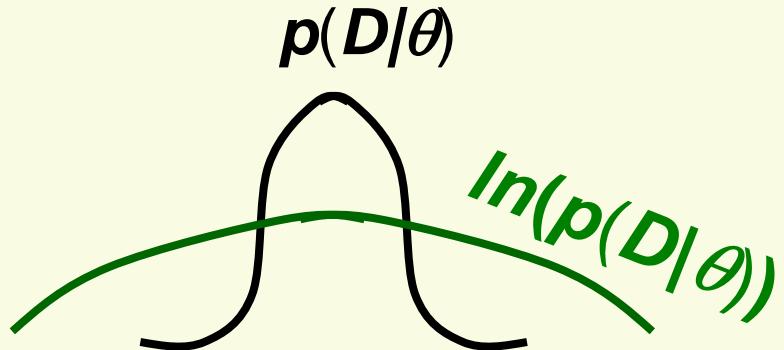
- ML classifier and ML parameter estimation use the same principles applied to different problems

Maximum Likelihood Estimation (MLE)

- Instead of maximizing $p(D|\theta)$, it is usually easier to maximize $\ln(p(D|\theta))$

- Since log is monotonic

$$\begin{aligned}\hat{\theta} &= \underset{\theta}{\operatorname{argmax}}(p(D|\theta)) = \\ &= \underset{\theta}{\operatorname{argmax}}(\ln p(D|\theta))\end{aligned}$$



- To simplify notation, $\ln(p(D|\theta))=L(\theta)$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} L(\theta) = \underset{\theta}{\operatorname{argmax}} \left(\ln \prod_{k=1}^{k=n} p(x_k | \theta) \right) = \underset{\theta}{\operatorname{argmax}} \left(\sum_{k=1}^n \ln p(x_k | \theta) \right)$$

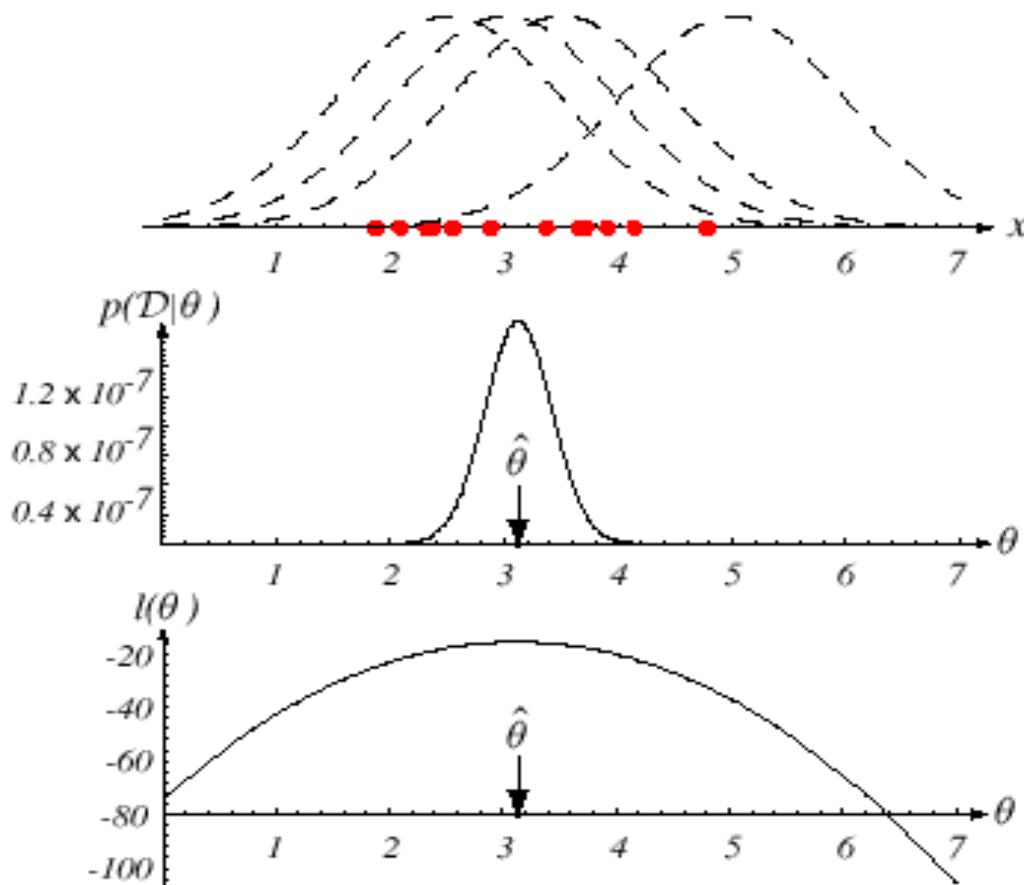


FIGURE 3.1. The top graph shows several training points in one dimension, known or assumed to be drawn from a Gaussian of a particular variance, but unknown mean. Four of the infinite number of candidate source distributions are shown in dashed lines. The middle figure shows the likelihood $p(\mathcal{D}|\theta)$ as a function of the mean. If we had a very large number of training points, this likelihood would be very narrow. The value that maximizes the likelihood is marked $\hat{\theta}$; it also maximizes the logarithm of the likelihood—that is, the log-likelihood $l(\theta)$, shown at the bottom. Note that even though they look similar, the likelihood $p(\mathcal{D}|\theta)$ is shown as a function of θ whereas the conditional density $p(x|\theta)$ is shown as a function of x . Furthermore, as a function of θ , the likelihood $p(\mathcal{D}|\theta)$ is not a probability density function and its area has no significance. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

MLE: Maximization Methods

- Maximizing $L(\theta)$ can be solved using standard methods from Calculus
- Let $\theta = (\theta_1, \theta_2, \dots, \theta_p)^t$ and let ∇_θ be the gradient operator

$$\nabla_\theta = \left[\frac{\partial}{\partial \theta_1}, \frac{\partial}{\partial \theta_2}, \dots, \frac{\partial}{\partial \theta_p} \right]^t$$

- Set of necessary conditions for an optimum is:

$$\nabla_\theta L = 0$$

- Also have to check that θ that satisfies the above condition is maximum, not minimum or saddle point. Also check the boundary of range of θ

MLE Example: Gaussian with unknown μ

- Fortunately for us, most of the ML estimates of any densities we would care about have been computed
- Let's go through an example anyway
- Let $p(x|\mu)$ be $N(\mu, \sigma^2)$ that is σ^2 is known, but μ is unknown and needs to be estimated, so $\theta = \mu$

$$\begin{aligned}\hat{\mu} &= \arg \max_{\mu} L(\mu) = \arg \max_{\mu} \left(\sum_{k=1}^n \ln p(x_k | \mu) \right) = \\ &= \arg \max_{\mu} \left(\sum_{k=1}^n \ln \left(\frac{1}{\sqrt{2\pi\sigma}} \exp \left(-\frac{(x_k - \mu)^2}{2\sigma^2} \right) \right) \right) = \\ &= \arg \max_{\mu} \sum_{k=1}^n \left(-\ln \sqrt{2\pi\sigma} - \frac{(x_k - \mu)^2}{2\sigma^2} \right)\end{aligned}$$

MLE Example: Gaussian with unknown μ

$$\arg \max_{\mu} (L(\mu)) = \arg \max_{\mu} \sum_{k=1}^n \left(-\ln \sqrt{2\pi\sigma^2} - \frac{(x_k - \mu)^2}{2\sigma^2} \right)$$

$$\begin{aligned} \frac{d}{d\mu} (L(\mu)) &= \sum_{k=1}^n \frac{1}{\sigma^2} (x_k - \mu) = 0 \Rightarrow \sum_{k=1}^n x_k - n\mu = 0 \Rightarrow \\ &\Rightarrow \hat{\mu} = \frac{1}{n} \sum_{k=1}^n x_k \end{aligned}$$

- Thus the ML estimate of the mean is just the average value of the training data, very intuitive!
 - average of the training data would be our guess for the mean even if we didn't know about ML estimates

MLE for Gaussian with unknown μ, σ^2

- Similarly it can be shown that if $p(\mathbf{x} | \mu, \sigma^2)$ is $\mathcal{N}(\mu, \sigma^2)$, that is both mean and variance are unknown, then again very intuitive result

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k \quad \hat{\sigma}^2 = \frac{1}{n} \sum_{k=1}^n (\mathbf{x}_k - \hat{\mu})^2$$

- Similarly it can be shown that if $p(\mathbf{x} | \mu, \Sigma)$ is $\mathcal{N}(\mu, \Sigma)$, that is \mathbf{x} is a multivariate Gaussian with both mean and covariance matrix unknown, then

$$\hat{\mu} = \frac{1}{n} \sum_{k=1}^n \mathbf{x}_k \quad \hat{\Sigma} = \frac{1}{n} \sum_{k=1}^n (\mathbf{x}_k - \hat{\mu})(\mathbf{x}_k - \hat{\mu})^t$$

How to Measure Performance of MLE?

- How good is a ML estimate $\hat{\theta}$?
 - or actually any other estimate of a parameter?
- The natural measure of error would be $|\theta - \hat{\theta}|$
- But $|\theta - \hat{\theta}|$ is random, we cannot compute it before we carry out experiments
 - We want to say something meaningful about our estimate as a function of θ
- A way to solve this difficulty is to **average** the error, i.e. compute the **mean absolute error**

$$E[|\theta - \hat{\theta}|] = \int |\theta - \hat{\theta}| p(x_1, x_2, \dots, x_n) dx_1 dx_2 \dots dx_n$$

How to Measure Performance of MLE's

- It is usually much easier to compute an almost equivalent measure of performance, the **mean squared error**:

$$E[(\theta - \hat{\theta})^2]$$

- Do a little algebra, and use $\text{Var}(X) = E(X^2) - (E(X))^2$

$$E[(\theta - \hat{\theta})^2] = \underbrace{\text{Var}(\hat{\theta})}_{\text{variance}}$$

$$+ \underbrace{(E(\hat{\theta}) - \theta)^2}_{\text{bias}}$$

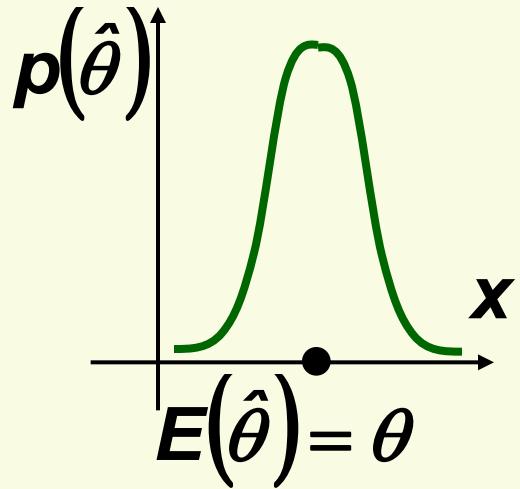
estimator should have low variance

expectation should be close to the true θ

How to Measure Performance of MLE?

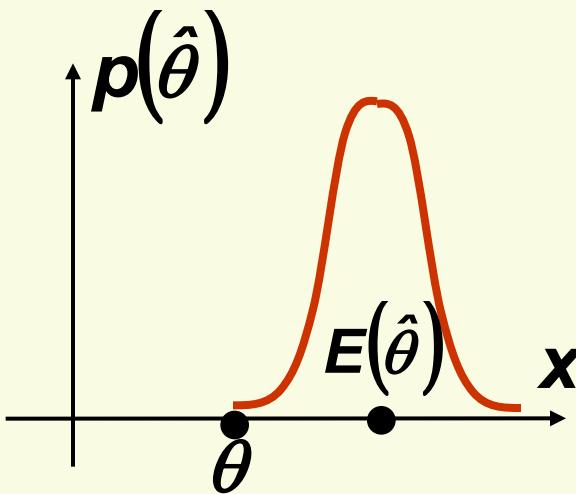
$$E[(\theta - \hat{\theta})^2] = \underbrace{Var(\hat{\theta})}_{\text{variance}} + \underbrace{(E(\hat{\theta}) - \theta)^2}_{\text{bias}}$$

ideal case



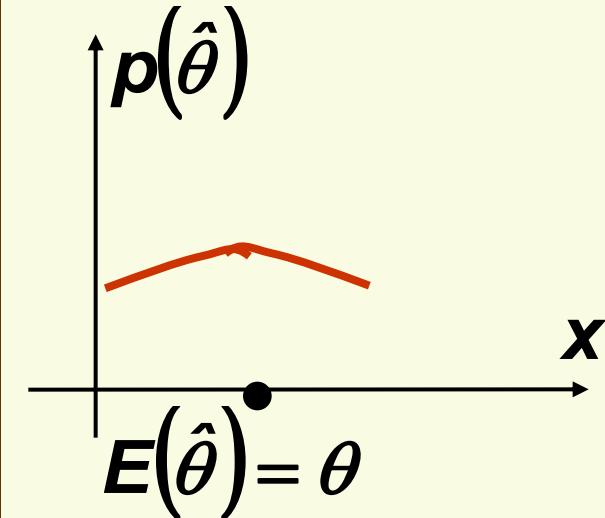
*no bias
low variance*

bad case



*large bias
low variance*

bad case



*no bias
high variance*

Bias and Variance for MLE of the Mean

- Let's compute the bias for ML estimate of the mean

$$E[\hat{\mu}] = E\left[\frac{1}{n} \sum_{k=1}^n x_k\right] = \frac{1}{n} \sum_{k=1}^n E[x_k] = \frac{1}{n} \sum_{k=1}^n \mu = \mu$$

- Thus this estimate is unbiased!
- How about variance of ML estimate of the mean?

$$\begin{aligned} E[(\hat{\mu} - \mu)^2] &= E\left[\left(\frac{1}{n} \sum_{i=1}^n x_i - \mu\right)^2\right] = E\left[\left(\frac{1}{n} \sum_{i=1}^n (x_i - \mu)\right)^2\right] \\ &= \frac{1}{n^2} E\left[\sum_{i=1}^n \sum_{j=1}^n (x_i - \mu)(x_j - \mu)\right] = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n E[(x_i - \mu)(x_j - \mu)] \\ &= \frac{1}{n^2} n \sigma^2 = \frac{\sigma^2}{n} \end{aligned}$$

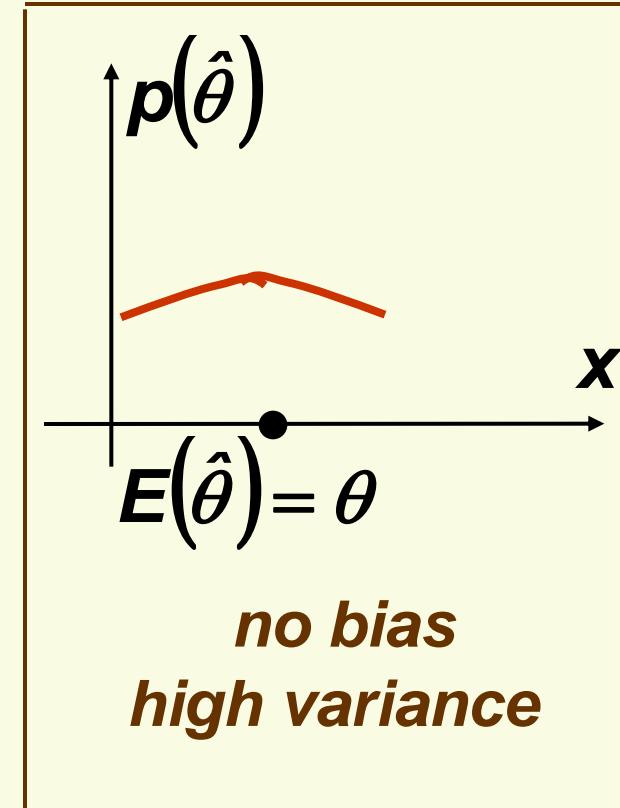
- Thus variance is very small for a large number of samples (the more samples, the smaller is variance)
- Thus the MLE of the mean is a very good estimator

Bias and Variance for MLE of the Mean

- Suppose someone claims they have a new great estimator for the mean, just take the first sample!

$$\hat{\mu} = x_1$$

- Thus this estimator is unbiased: $E(\hat{\mu}) = E(x_1) = \mu$
- However its variance is:
$$E[(\hat{\mu} - \mu)^2] = E[(x_1 - \mu)^2] = \sigma^2$$
- Thus variance can be very large and does not improve as we increase the number of samples



MLE Bias for Mean and Variance

- How about ML estimate for the variance?

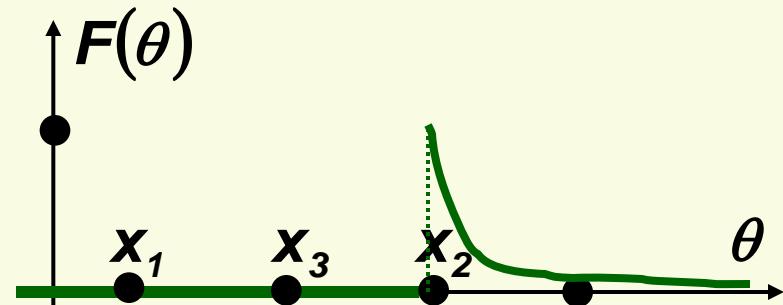
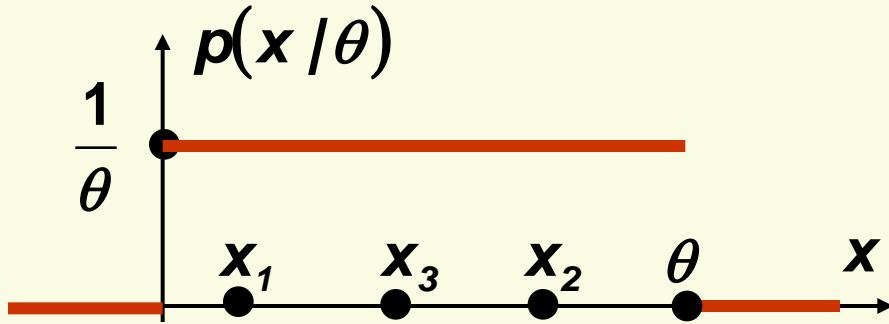
$$E[\hat{\sigma}^2] = E\left[\frac{1}{n} \sum_{k=1}^n (x_k - \hat{\mu})^2\right] = \frac{n-1}{n} \sigma^2 \neq \sigma^2$$

See http://en.wikipedia.org/wiki/Bias_of_an_estimator for details.

- Thus this estimate is biased!
 - This is because we used $\hat{\mu}$ instead of true μ
- Bias $\rightarrow 0$ as $n \rightarrow \infty$, *asymptotically* unbiased
- Unbiased estimate $\hat{\sigma}^2 = \frac{1}{n-1} \sum_{k=1}^n (x_k - \hat{\mu})^2$
- Variance of MLE of variance can be shown to go to 0 as n goes to infinity

MLE for Uniform distribution $U[0, \theta]$

- X is $U[0, \theta]$ if its density is $1/\theta$ inside $[0, \theta]$ and 0 otherwise (uniform distribution on $[0, \theta]$)



- The likelihood is $F(\theta) = \prod_{k=1}^{n=k} p(x_k | \theta) = \begin{cases} \frac{1}{\theta^n} & \text{if } \theta \geq \max\{x_1, \dots, x_n\} \\ 0 & \text{if } \theta < \max\{x_1, \dots, x_n\} \end{cases}$
- Thus $\hat{\theta} = \operatorname{argmax}_{\theta} \left(\prod_{k=1}^{n=k} p(x_k | \theta) \right) = \max\{x_1, \dots, x_n\}$
- This is not very pleasing since for sure θ should be larger than any observed x !

MLE Tutorial

C6

Problem 1

Show that if our model is poor, the maximum likelihood classifier we derive is not the best— even among our (poor) model set— by exploring the following example.

Suppose we have two equally probable categories (i.e., $P(\omega_1) = P(\omega_2) = 0.5$). Further, we know that $p(x|\omega_1) \sim N(0, 1)$ but assume that $p(x|\omega_2) \sim N(\mu, 1)$. (That is, the parameter θ we seek by maximum-likelihood techniques is the mean of the second distribution.) Imagine, however, that the true underlying distribution is $p(x|\omega_2) \sim N(1, 10^6)$.

$$p(x|\omega_1) \sim N(0, 1); p(x|\omega_2) \sim N(\mu, 1). \\ (p(x|\omega_2) \sim N(1, 10^6).)$$

- What is the value of our maximum-likelihood estimate $\hat{\mu}$ in our poor model, given a large amount of data?

$$\hat{\mu} = 1$$

- What is the decision boundary arising from this maximum-likelihood estimate in the poor model?

-> $p(x|\omega_1) \sim N(0, 1)$, $p(x|\omega_2) \sim N(1, 1)$,
 $P(\omega_1) = P(\omega_2) = 0.5$,
The decision boundary is $x=0.5$

- Ignore for the moment the maximum-likelihood approach, and derive the Bayes optimal decision boundary given the true underlying distributions: $p(x|\omega_1) \sim N(0, 1)$ and $p(x|\omega_2) \sim N(1, 10^6)$. Be careful to include all portions of the decision boundary.
-> The decision boundary is given by x that satisfies: $P(\omega_1) p(x|\omega_1) = P(\omega_2) p(x|\omega_2)$
Plug in the models and take ln of both sides:

$$-\frac{x^2}{2} = -\frac{(x-1)^2}{2 \times 10^6} - \ln 10^3$$

$$x_1 = -3.7169, \quad x_2 = 3.7169$$

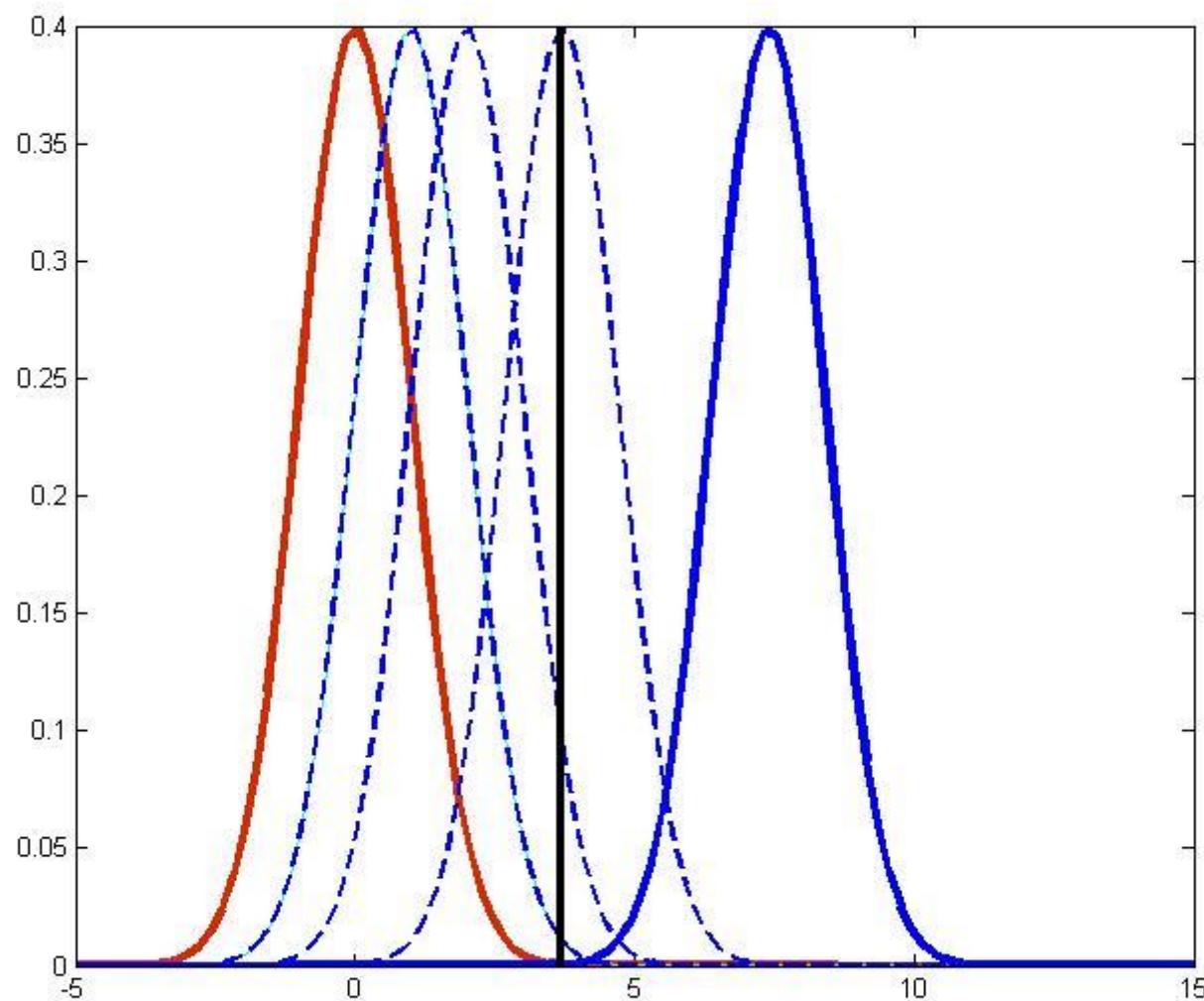
The decision regions

for ω_1 is $[-3.7169, 3.7169]$

for ω_2 is $[-\infty, -3.7169] \cup [3.7169, \infty]$

- Now consider again classifiers based on the (poor) model assumption of $p(x|\omega_2) \sim N(\mu, 1)$. Using your result, find a new value of μ that will give lower error than the maximum-likelihood classifier.

->The decision according to the poor model is not optimal in $[0.5, 3.7169]$. By moving the decision toward 3.7169 we can reduce the error. Thus any $1 < \mu < 7.4338$ will reduce the error.



- Discuss these results, with particular attention to the role of knowledge of the underlying model.

->This example shows that the parametric form is very important for maximum-likelihood estimation. If the assumed form is far from the true underlying model, the ML estimate can give larger error than other models in the same assumed family. In order to get good results using ML estimate, one needs to find the most accurate model for the unknown underlying model based on prior knowledge, experience, or experiments on some data. If several models are available, they should be evaluated and compared using some test data.

Problem 2

- Maximum likelihood methods apply to estimates of prior probability as well. Let samples been drawn by successive independent selection of state of nature ω_i with unknown probability $P(\omega_i)$. Let

$$z_{ik} = \begin{cases} 1 & \text{if the } k^{\text{th}} \text{ sample belongs to } \omega_i \\ 0 & \text{otherwise} \end{cases}$$

- a) Show that

$$P(z_{i1}, \dots, z_{in} | P(\omega_i)) = \prod_{k=1}^n P(\omega_i)^{z_{ik}} (1 - P(\omega_i))^{1-z_{ik}}$$

- b) Show the MLE for $P(\omega_i)$ is $\hat{P}(\omega_i) = \frac{1}{n} \sum_{i=1}^n z_{ik}$

Interpret your results in words.

a) Solution: $P(z_{ik} = 1 | P(\omega_i)) = P(\omega_i), P(z_{ik} = 0 | P(\omega_i)) = 1 - P(\omega_i)$

Combining them together we have

$$P(z_{ik} | P(\omega_i)) = P(\omega_i)^{z_{ik}} (1 - P(\omega_i))^{1-z_{ik}}$$

z_{i1}, \dots, z_{in} are drawn independently, thus

$$P(z_{i1}, \dots, z_{in} | P(\omega_i)) = \prod_{k=1}^n P(z_{ik} | \omega_i) = \prod_{k=1}^n P(\omega_i)^{z_{ik}} (1 - P(\omega_i))^{1-z_{ik}}$$

b) Solution:

$$\ln P(z_{i1}, \dots, z_{in} | P(\omega_i)) = \sum_{k=1}^n (z_{ik} \ln P(\omega_i) + (1 - z_{ik}) \ln(1 - P(\omega_i)))$$

$$\frac{\partial \ln P(z_{i1}, \dots, z_{in} | P(\omega_i))}{\partial P(\omega_i)} = \sum_{k=1}^n \left(z_{ik} \frac{1}{P(\omega_i)} - (1 - z_{ik}) \frac{1}{1 - P(\omega_i)} \right) = 0$$

$$(1 - P(\omega_i)) \left(\sum z_{ik} \right) - P(\omega_i) \left(\sum (1 - z_{ik}) \right) = 0$$

$$\hat{P}(\omega_i) = \frac{1}{n} \sum_{i=1}^n z_{ik}$$

$\hat{P}(\omega_i)$ is the fraction of samples from class i among all of the samples.

Problem 3

- Write down the training steps and a set of discriminant functions for minimum error rate classification. You need to include the details.
- First we fix the notation, let $x_{i,k}^j$ represent the k^{th} feature in the i^{th} example of class ω_j . Assume that we have n training samples in total and we have n_1 for ω_1 , ..., and n_C for ω_C

The discriminant functions for minimum error rate classification.

$$g_j(x) = \ln p(x | \omega_j) + \ln P(\omega_j)$$

Here for class j , we need to estimate both $P(\omega_j)$ and $p(x | \omega_j)$.
For $P(\omega_j)$, using maximum likelihood estimation, we have

$$P(\omega_j) = \frac{n_j}{n}$$

To estimate $p(x | \omega_j)$, based on the assumptions that the features are statistically independent and they are normally distributed, we have

$$p(x | \omega_j) = \prod_{k=1}^d \frac{1}{\sqrt{2\pi}\sigma_{j,k}} e^{-\frac{(x_k - \mu_{j,k})^2}{2\sigma_{j,k}^2}}$$

Here class ω_j has 2d parameters and they are estimated according to maximum likelihood estimation

$$\mu_{j,k} = \frac{1}{n_j} \sum_{i=1}^{n_j} x_{i,k}^j$$

$$\sigma^2_{j,k} = \frac{1}{n_j} \sum_{i=1}^{n_j} (x_{i,k}^j - \mu_{j,k})^2$$

Plug-in the results to the discriminant function given above and ignore the common constant, we have

$$g_j(x) = - \left(\sum_{k=1}^d \left(\ln \sigma^2_{j,k} + \frac{(x_k - \mu_{j,k})^2}{\sigma^2_{j,k}} \right) \right) + \ln n_j$$

Parametric Density Estimation:

Bayesian Estimation.

Naïve Bayes Classifier

Bayesian Parameter Estimation

- Suppose we have some idea of the range where parameters θ should be
 - Shouldn't we formalize such prior knowledge in hopes that it will lead to better parameter estimation?
- Let θ be a random variable with prior distribution $P(\theta)$
 - This is the key difference between ML and Bayesian parameter estimation
 - This key assumption allows us to fully exploit the information provided by the data

Bayesian Parameter Estimation

- θ is a random variable with prior $p(\theta)$
 - Unlike MLE case, $p(x|\theta)$ is a conditional density
- The training data D allow us to convert $p(\theta)$ to a posterior probability density $p(\theta|D)$.
 - After we observe the data D, using Bayes rule we can compute the posterior $p(\theta|D)$
- But θ is not our final goal, our final goal is the unknown $p(x)$
- Therefore a better thing to do is to maximize $p(x|D)$, this is as close as we can come to the unknown $p(x)$!

Bayesian Estimation: Formula for $p(x|D)$

- From the definition of joint distribution:

$$p(x|D) = \int p(x, \theta | D) d\theta$$

- Using the definition of conditional probability:

$$p(x|D) = \int p(x|\theta, D)p(\theta|D)d\theta$$

- But $p(x|\theta, D) = p(x|\theta)$ since $p(x|\theta)$ is completely specified by θ

$$p(x|D) = \int \begin{matrix} \text{known} \\ p(x|\theta) \end{matrix} \begin{matrix} \text{unknown} \\ p(\theta|D) \end{matrix} d\theta$$

- Using Bayes formula,

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)d\theta}$$

$$p(D|\theta) = \prod_{k=1}^n p(x_k|\theta)$$

Bayesian Estimation vs. MLE

- So in principle $p(x|D)$ can be computed
 - In practice, it may be hard to do integration analytically, may have to resort to numerical methods

$$p(x|D) = \int p(x|\theta) \frac{\prod_{k=1}^n p(x_k|\theta)p(\theta)}{\int \prod_{k=1}^n p(x_k|\theta)p(\theta)d\theta} d\theta$$

- Contrast this with the MLE solution which requires differentiation of likelihood to get $p(x|\hat{\theta})$
 - Differentiation is easy and can always be done analytically

Bayesian Estimation vs. MLE

$$p(x | D) = \int p(x | \theta) \underbrace{p(\theta | D)}_{\text{proposed model with certain } \theta} d\theta$$

support θ receives from the data

- The above equation implies that if we are less certain about the exact value of θ , we should consider a weighted average of $p(x|\theta)$ over the possible values of θ .
- Contrast this with the MLE solution which always gives us a single model:

$$p(x | \hat{\theta})$$

Bayesian Estimation for Gaussian with unknown μ

- Let $p(x|\mu)$ be $N(\mu, \sigma^2)$ that is σ^2 is known, but μ is unknown and needs to be estimated, so $\theta = \mu$
- Assume a prior over μ : $p(\mu) \sim N(\mu_0, \sigma_0^2)$
- μ_0 encodes some prior knowledge about the true mean μ , while σ_0^2 measures our prior uncertainty.
- The posterior distribution is:

$$p(\mu | D) \propto p(D | \mu) p(\mu)$$

factors that
do not
depend on μ

$$\begin{aligned} &= \alpha' \exp \left[-\frac{1}{2} \left(\sum_{k=1}^n \left(\frac{x_k - \mu}{\sigma} \right)^2 + \left(\frac{\mu - \mu_0}{\sigma_0} \right)^2 \right) \right] \\ &= \alpha'' \exp \left[-\frac{1}{2} \left[\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \right) \mu^2 - 2 \left(\frac{1}{\sigma^2} \sum_{k=1}^n x_k + \frac{\mu_0}{\sigma_0^2} \right) \mu \right] \right] \end{aligned}$$

Bayesian Estimation for Gaussian with unknown μ

- $p(\mu | D)$ remains normal for any number of training samples.
 - The property that the posterior distribution follows the same parametric form as the prior distribution is called **conjugacy**
 - the prior distribution is called a **conjugate prior**

- If we write
$$p(\mu | D) = \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left[-\frac{1}{2}\left(\frac{\mu - \mu_n}{\sigma_n}\right)^2\right]$$

$$\alpha'' \exp\left[-\frac{1}{2}\left[\left(\frac{n}{\sigma^2} + \frac{1}{\sigma_0^2}\right)\mu^2 - 2\left(\frac{1}{\sigma^2} \sum_{k=1}^n x_k + \frac{\mu_0}{\sigma_0^2}\right)\mu\right]\right]$$

Bayesian Estimation for Gaussian with unknown μ

- then identifying the coefficients, we get

$$\frac{1}{\sigma_n^2} = \frac{n}{\sigma^2} + \frac{1}{\sigma_0^2} \quad \frac{\mu_n}{\sigma_n^2} = \frac{n}{\sigma^2} \hat{\mu}_n + \frac{\mu_0}{\sigma_0^2}$$

where $\hat{\mu}_n = \frac{1}{n} \sum_{k=1}^n x_k$ is the sample mean

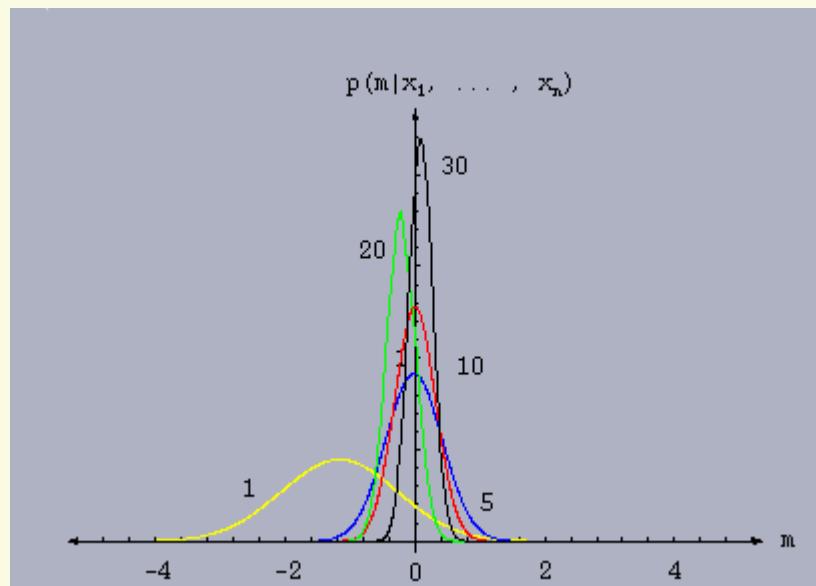
- Solving explicitly for μ_n and σ_n^2 we obtain:

$$\mu_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2} \right) \hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2} \mu_0 \text{ our best guess after observing n samples}$$

$$\sigma_n^2 = \frac{\sigma_0^2 \sigma^2}{n\sigma_0^2 + \sigma^2} \quad \text{uncertainty about the guess, decreases monotonically with n}$$

Bayesian Estimation for Gaussian with unknown μ

- Each additional observation decreases our uncertainty about the true value of μ .
- As n increases, $p(\mu | D)$ becomes more and more sharply peaked, approaching a Dirac delta function as n approaches infinity. This behavior is known as Bayesian Learning.



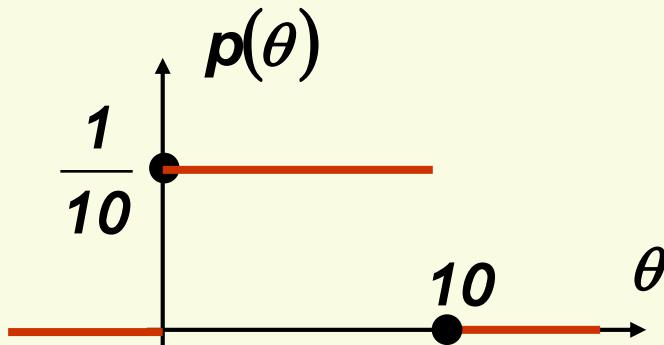
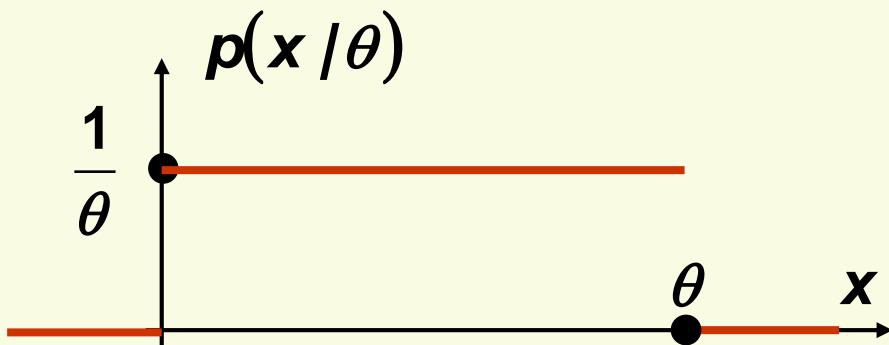
Bayesian Estimation for Gaussian with unknown μ

$$\mu_n = \left(\frac{n\sigma_0^2}{n\sigma_0^2 + \sigma^2} \right) \hat{\mu}_n + \frac{\sigma^2}{n\sigma_0^2 + \sigma^2} \mu_0$$

- In general, μ_n is a linear combination of a **sample mean** $\hat{\mu}_n$ and a **prior** μ_0 , with coefficients that are non-negative and sum to 1.
- Thus μ_n lies somewhere between $\hat{\mu}_n$ and μ_0 .
- If $\sigma_0 \neq 0$, $\mu_n \rightarrow \hat{\mu}_n$ as $n \rightarrow \infty$
- If $\sigma_0 = 0$, our a priori certainty that $\mu = \mu_0$ is so strong that no number of observations can change our opinion.
- If a priori guess is very uncertain (σ_0 is large), we take $\mu_n = \hat{\mu}_n$

Bayesian Estimation: Example for $U[0, \theta]$

- Let X be $U[0, \theta]$. Recall $p(x|\theta) = 1/\theta$ inside $[0, \theta]$, else 0



- Suppose we assume a $U[0, 10]$ prior on θ
 - good prior to use if we just know the range of θ but don't know anything else

Bayesian Estimation: Example for $U[0, \theta]$

- We need to compute $p(x | D) = \int p(x | \theta)p(\theta | D)d\theta$
- using $p(\theta | D) = \frac{p(D | \theta)p(\theta)}{\int p(D | \theta)p(\theta)d\theta}$ and $p(D | \theta) = \prod_{k=1}^n p(x_k | \theta)$
- When computing MLE of θ , we had

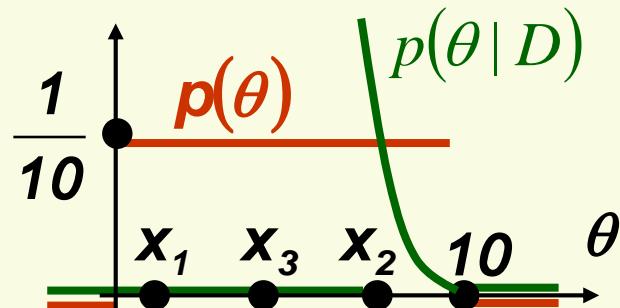
$$p(D | \theta) = \begin{cases} \frac{1}{\theta^n} & \text{for } \theta \geq \max\{x_1, \dots, x_n\} \\ 0 & \text{otherwise} \end{cases}$$

- Thus

$$p(\theta | D) = \begin{cases} c \frac{1}{\theta^n} & \text{for } \max\{x_1, \dots, x_n\} \leq \theta \leq 10 \\ 0 & \text{otherwise} \end{cases}$$

- where c is the normalizing constant, i.e.

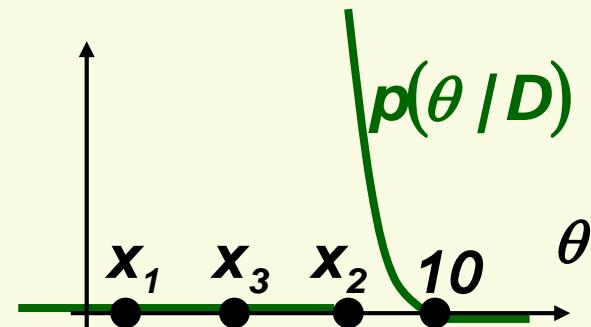
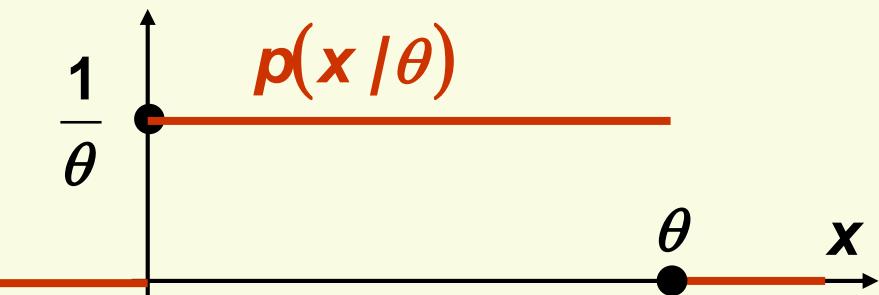
$$c = \frac{1}{\int_{\max\{x_1, \dots, x_n\}}^{10} \frac{d\theta}{\theta^n}}$$



Bayesian Estimation: Example for $U[0, \theta]$

- We need to compute $p(x | D) = \int p(x | \theta)p(\theta | D)d\theta$

$$p(\theta | D) = \begin{cases} C \frac{1}{\theta^n} & \text{for } \max\{x_1, \dots, x_n\} \leq \theta \leq 10 \\ 0 & \text{otherwise} \end{cases}$$



- We have 2 cases:

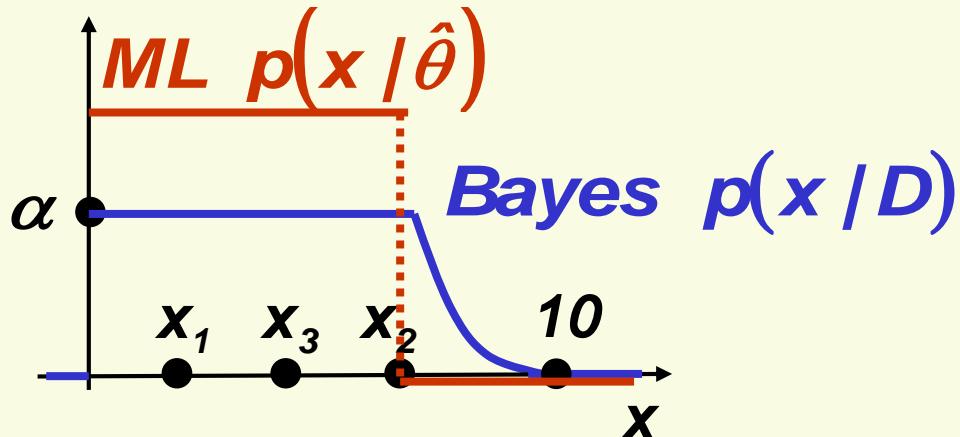
- case $x < \max\{x_1, x_2, \dots, x_n\}$

$$p(x | D) = \int_{\max\{x_1, \dots, x_n\}}^{10} C \frac{1}{\theta^{n+1}} d\theta = \boxed{\alpha} \quad \begin{matrix} \text{constant} \\ \text{independent of } x \end{matrix}$$

- case $x > \max\{x_1, x_2, \dots, x_n\}$

$$p(x | D) = \int_x^{10} C \frac{1}{\theta^{n+1}} d\theta = \left. -\frac{C}{n\theta^n} \right|_x^{10} = \boxed{\frac{C}{nx^n}} - \frac{C}{n10^n}$$

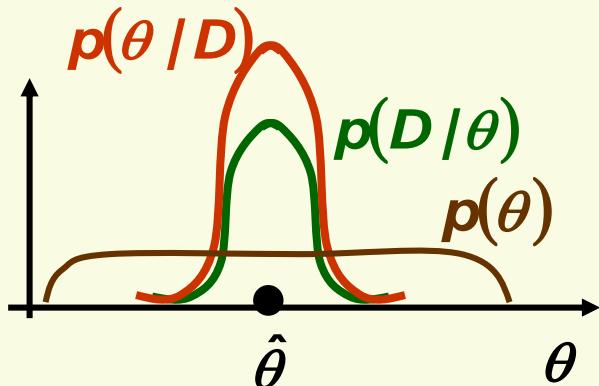
Bayesian Estimation: Example for $U[0, \theta]$



- Note that even after $x > \max \{x_1, x_2, \dots, x_n\}$, Bayes density is not zero, which makes sense
- curious fact: Bayes density is not uniform, i.e. does not have the functional form that we have assumed!

ML vs. Bayesian Estimation with Broad Prior

- Suppose $p(\theta)$ is flat and broad (close to uniform prior)
- $p(\theta|D)$ tends to sharpen if there is a lot of data



- Thus $p(D|\theta) \propto p(\theta|D)p(\theta)$ will have the same sharp peak as $p(\theta|D)$
- But by definition, peak of $p(D|\theta)$ is the ML estimate $\hat{\theta}$
- The integral is dominated by the peak:

$$p(x|D) = \int p(x|\theta)p(\theta|D)d\theta \approx p(x|\hat{\theta}) \int p(\theta|D)d\theta = p(x|\hat{\theta})$$

- Thus as n goes to infinity, Bayesian estimate will approach the density corresponding to the MLE!

ML vs. Bayesian Estimation

- **Number of training data**
 - The two methods are equivalent assuming infinite number of training data (and prior distributions that do not exclude the true solution).
 - For small training data sets, they give different results in most cases.
- **Computational complexity**
 - ML uses differential calculus or gradient search for maximizing the likelihood.
 - Bayesian estimation requires complex multidimensional integration techniques.

ML vs. Bayesian Estimation

- **Solution complexity**
 - Easier to interpret ML solutions (i.e., must be of the assumed parametric form).
 - A Bayesian estimation solution might not be of the parametric form assumed. Hard to interpret, returns weighted average of models.
- **Broad or asymmetric $p(\theta|D)$**
 - In this case, the two methods will give different solutions.
 - Bayesian methods will explicitly exploit such information.

ML vs. Bayesian Estimation

- **General comments**

- There are strong theoretical and methodological arguments supporting Bayesian estimation.
- In practice, ML estimation is simpler and can lead to comparable performance.

Naïve Bayes Classifier

Unbiased Learning of Bayes Classifiers is Impractical

- Learn Bayes classifier by estimating $P(X/Y)$ and $P(Y)$.
- Assume Y is boolean and X is a vector of n boolean attributes. In this case, we need to estimate a set of parameters $\theta_{ij} \equiv P(X = x_i | Y = y_j)$
i takes on 2^n possible values; *j* takes on 2 possible values.
- How many parameters?
 - For any particular value y_j , and the 2^n possible values of x_i , we need compute 2^n-1 independent parameters.
 - Given the two possible values for Y , we must estimate a total of $2(2^n-1)$ such parameters.

Complex model → High variance with limited data!!!

Conditional Independence

- **Definition:** X is **conditionally independent** of Y given Z, if the probability distribution governing X is independent of the value of Y, given the value of Z

$$P(X, Y | Z) = P(X | Y, Z)P(Y | Z) = P(X | Z)P(Y | Z)$$

Derivation of Naive Bayes Algorithm

- **Naive Bayes algorithm** assumes that the attributes X_1, \dots, X_n are all conditionally independent of one another, given Y . This dramatically simplifies
 - the representation of $P(X|Y)$
 - estimating $P(X|Y)$ from the training data.
- Consider $X=(X_1, X_2)$

$$P(X | Y) = P(X_1, X_2 | Y) = P(X_1 | Y)P(X_2 | Y)$$

- For X containing n attributes

$$P(X | Y) = \prod_{i=1}^n P(X_i | Y)$$

Given the boolean X and Y , now we need only $2n$ parameters to define $P(X|Y)$, which is dramatic reduction compared to the $2(2^n - 1)$ parameters if we make no conditional independence assumption.

The Naïve Bayes Classifier

- Given:
 - Prior $P(Y)$
 - n conditionally independent features X , given the class Y
 - For each X_i , we have likelihood $P(X_i|Y)$
- The probability that Y will take on its k^{th} possible value, is

$$P(Y = y_k | X) = \frac{P(Y = y_k) \prod_i P(X_i | Y = y_k)}{\sum_j P(Y = y_k) \prod_i P(X_i | Y = y_k)}$$

- The Decision rule:

$$y^* = \arg \max_{y_k} P(Y = y_k) \prod_i P(X_i | Y = y_k)$$

If assumption holds, NB is optimal classifier!

Naïve Bayes for the discrete inputs

- Given, n attributes X_i each taking on J possible discrete values and Y a discrete variable taking on K possible values.
- MLE for Likelihood $P(X_i = x_{ij} | Y = y_k)$ given a set of training examples D :

$$\hat{P}(X_i = x_{ij} | Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\}}{\#D\{Y = y_k\}}$$

where the $\#D\{x\}$ operator returns the number of elements in the set D that satisfy property x .

- MLE for the prior

$$\hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\}}{|D|}$$

← number of elements
in the training set D

NB Example

- Given, training data

X

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Y

- Classify the following novel instance :
(Outlook=sunny, Temp=cool, Humidity=high, Wind=strong)

NB Example

$$y_{NB} = \arg \max_{y_j=\{yes,no\}} P(y_j)P(Outlook = sunny | y_j)P(Temp = cool | y_j)$$

$$P(Humidity = high | y_j)P(Wind = strong | y_j)$$

Priors :

$$P(PlayTennis = yes) = 9/14 = 0.64$$

$$P(PlayTennis = no) = 5/14 = 0.36$$

Conditional Probabilities, e.g. Wind = strong:

$$P(Wind = strong | PlayTennis = yes) = 3/9 = 0.33$$

$$P(Wind = strong | PlayTennis = no) = 3/5 = 0.6$$

...

$$P(yes)P(sunny | yes)P(cool | yes)P(high | yes)P(strong | yes) = 0.0053$$

$$P(no)P(sunny | no)P(cool | no)P(high | no)P(strong | no) = 0.60$$

Subtleties of NB classifier 1 – Violating the NB assumption

- Usually, features are not conditionally independent.
- Nonetheless, NB often performs well, even when assumption is violated
 - [Domingos& Pazzani'96] discuss some conditions for good performance

Subtleties of NB classifier 2 – Insufficient training data

- What if you never see a training instance where $X_1=a$ when $Y=b$?
 - $P(X_1=a | Y=b) = 0$
- Thus, no matter what the values X_2, \dots, X_n take:
$$P(Y=b | X_1=a, X_2, \dots, X_n) = 0$$
- Solution?

Subtleties of NB classifier 2 – Insufficient training data

- To avoid this, use a “smoothed” estimate
 - effectively adds in a number of additional “hallucinated” examples
 - assumes these hallucinated examples are spread evenly over the possible values of X_i .
- This smoothed estimate is given by

$$\hat{P}(X_i = x_{ij} \mid Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\} + l}{\#D\{Y = y_k\} + lJ}$$
$$\hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\} + l}{|D| + lK}$$

The number of
hallucinated examples

l determines the strength of the smoothing
If $l=1$ called Laplace smoothing

Naïve Bayes for Continuous Inputs

- When the X_i are continuous we must choose some other way to represent the distributions $P(X_i|Y)$.
- One common approach is to assume that for each possible discrete value y_k of Y , the distribution of each continuous X_i is Gaussian.
- In order to train such a Naïve Bayes classifier we must estimate the mean and standard deviation of each of these Gaussians

Naive Bayes for Continuous Inputs

- MLE for means

$$\hat{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k)$$

- where j refers to the j th training example, and where $\delta(Y=y_k)$ is 1 if $Y = y_k$ and 0 otherwise.
- Note the role of δ is to select only those training examples for which $Y = y_k$.

- MLE for standard deviation

$$\hat{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j (X_i^j - \hat{\mu}_{ik})^2 \delta(Y^j = y_k)$$

Learning Classify Text

- Applications:
 - Learn which news article are of interest
 - Learn to classify web pages by topic.
- Naïve Bayes is among most effective algorithms
- Target concept *Interesting?*: $Document \rightarrow \{+, -\}$
 - 1 Represent each document by vector of words
 - one attribute per word position in document
 - 2 Learning: Use training examples to estimate
 - $P(+)$
 - $P(-)$
 - $P(\text{doc}|+)$
 - $P(\text{doc}|-)$

Text Classification-Example:

Text

Text Classification, or the task of automatically assigning semantic categories to natural language text, has become one of the key methods for organizing online information. Since hand-coding classification rules is costly or even impractical, most modern approaches employ machine learning techniques to automatically learn text classifiers from examples.

The text contains 48 words

Text Representation

($a_1 = \text{'text'}$, $a_2 = \text{'classification'}$, ..., $a_{48} = \text{'examples'}$)



The representation
contains 48 attributes

Note: Text size may vary, but it will not cause a problem

NB conditional independence assumption

$$P(doc | y_j) = \prod_{i=1}^{\text{length}(doc)} P(a_i = w_k | y_j)$$

Indicates the kth word in English vocabulary

The NB assumption is that the word probabilities for one text position are independent of the words in other positions, given the document classification y_j

Clearly not true: The probability of word “learning” may be greater if the preceding word is “machine”

probability that word in position i is w_k , given y_j

Necessary, without it the number of probability terms is prohibitive

Performs remarkably well despite the incorrectness of the assumption

Text Classification-Example:

Text

Text Classification, or the task of automatically assigning semantic categories to natural language text, has become one of the key methods for organizing online information. Since hand-coding classification rules is costly or even impractical, most modern approaches employ machine learning techniques to automatically learn text classifiers from examples.

The text contains 48 words

Text Representation

($a_1 = \text{'text'}$, $a_2 = \text{'classification'}$, ..., $a_{48} = \text{'examples'}$)



The representation contains 48 attributes

Classification:

$$y^* = \arg \max_{y_j \in \{+, -\}} P(y_j) P(a_1 = \text{'text'} | y_j) \dots P(a_{48} = \text{'examples'} | y_j)$$

$$= \arg \max_{y_j \in \{+, -\}} P(y_j) \prod_i P(a_i = w_k | y_j)$$

Estimating Likelihood

- Is problematic because we need to estimate it for each combination of text position, English word, and target value: $48 * 50,000 * 2 \approx 5$ million such terms.
- Assumption that reduced the number of terms –
Bag of Words Model

- The probability of encountering a specific word w_k is independent of the specific word position.

$$P(a_i = w_k | y_j) = P(a_m = w_k | y_j), \quad \forall i, m$$

- Instead of estimating $P(a_1 = w_k | y_j)$, $P(a_k = w_k | y_j)$, ... we estimate a single term $P(w_k | y_j)$
- Now we have $50,000 * 2$ distinct terms.

Estimating Likelihood

- The estimate for the likelihood is

$$P(w_k \mid y_j) = \frac{n_k + 1}{n + |\text{Vocabulary}|}$$

n -the total number of word positions in all training examples whose target value is y_j ,
 n_k -the number times word w_k is found among these n word positions.
 $|\text{Vocabulary}|$ -the total number of distinct words found within the training data.

LEARN_NAIVE_BAYES_TEXT(*Examples*, V)

1. collect all words and other tokens that occur in *Examples*

- $\text{Vocabulary} \leftarrow$ all distinct words and other tokens in *Examples*

2. calculate the required $P(v_j)$ and $P(w_k|v_j)$ probability terms

- For each target value v_j in V do

- $\text{docs}_j \leftarrow$ subset of *Examples* for which the target value is v_j
- $P(v_j) \leftarrow \frac{|\text{docs}_j|}{|\text{Examples}|}$
- $\text{Text}_j \leftarrow$ a single document created by concatenating all members of docs_j
- $n \leftarrow$ total number of words in Text_j (counting duplicate words multiple times)
- for each word w_k in *Vocabulary*
 - * $n_k \leftarrow$ number of times word w_k occurs in Text_j
 - * $P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|\text{Vocabulary}|}$

Classify_Naive_Bayes_Text(*Doc*)

- *positions* \leftarrow all word positions in *Doc* that contain tokens found in *Vocabulary*
- Return $y^* = \arg \max_{y_j \in \{+, -\}} P(v_j) \prod_{i \in positions} P(w_i | v_j)$

Nonparametric Density Estimation

Intro

Parzen Windows

Non-Parametric Methods

- Neither probability distribution nor discriminant function is known
 - Happens quite often
- All we have is labeled data



- Estimate the probability distribution from the labeled data

*a lot is
known
"easier"*

*little is
known
"harder"*

NonParametric Techniques: Introduction

- In previous lectures we assumed that either
 1. someone gives us the density $p(\mathbf{x}/\mathbf{c}_j)$
 - In pattern recognition applications this never happens
 2. someone gives us $p(\mathbf{x}/\theta_{cj})$
 - Does happen sometimes, **but**
 - we are likely to suspect whether the given $p(\mathbf{x}/\theta)$ models the data well
 - Most parametric densities are unimodal (have a single local maximum), whereas many practical problems involve multi-modal densities

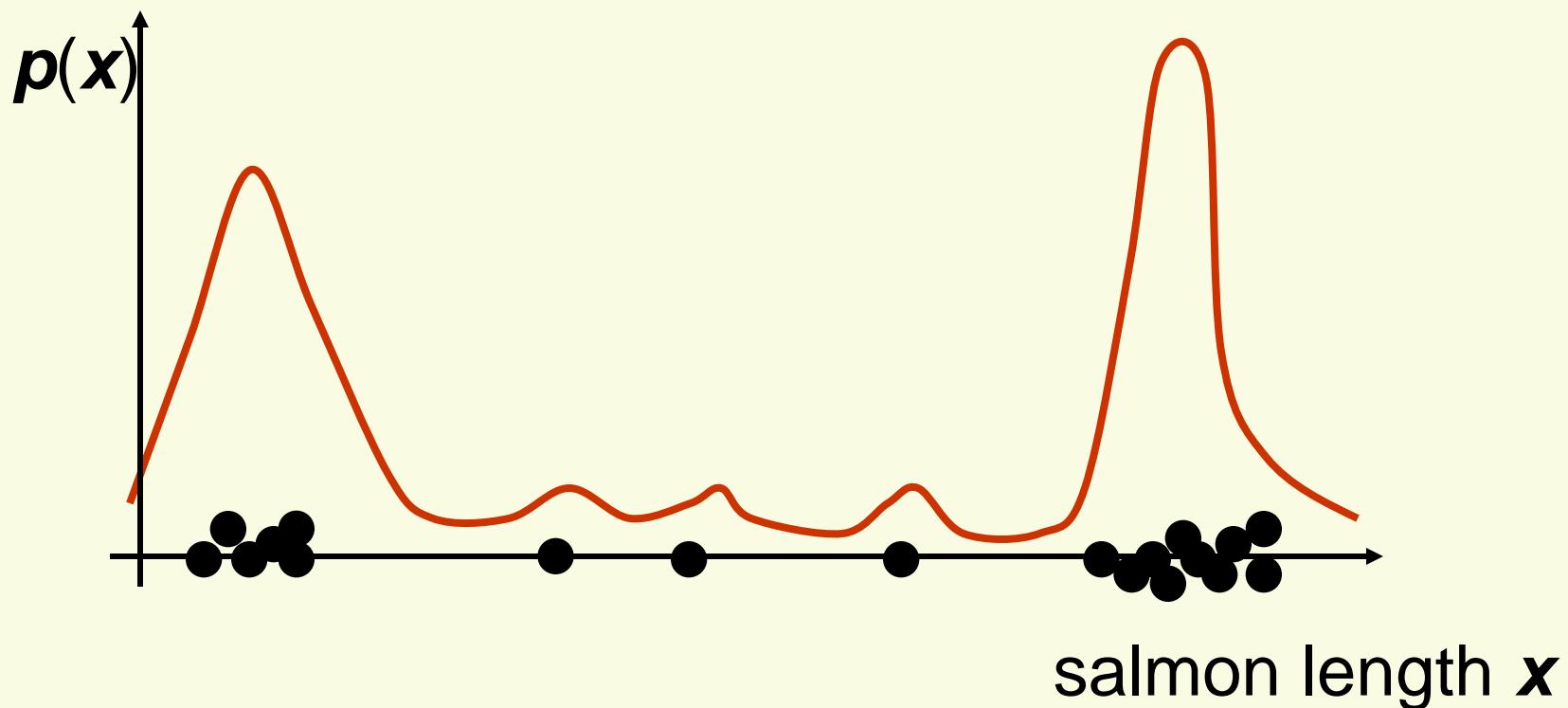
NonParametric Techniques: Introduction

- Nonparametric procedures can be used with arbitrary distributions and without any assumption about the forms of the underlying densities
- There are two types of nonparametric methods:
 - Parzen windows
 - Estimate likelihood $p(x | c_j)$
 - Nearest Neighbors
 - Bypass likelihood and go directly to posterior estimation $P(c_j | x)$

NonParametric Techniques: Introduction

- Nonparametric techniques attempt to estimate the underlying density functions from the training data
 - Idea: the more data in a region, the larger is the density function

$$Pr[X \in \mathcal{R}] = \int_{\mathcal{R}} f(x) dx$$



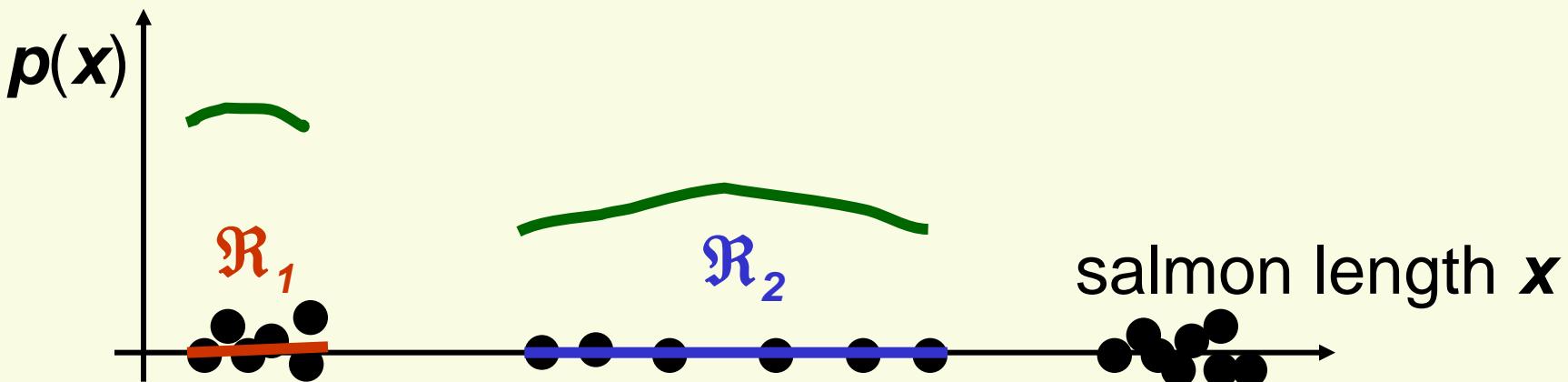
NonParametric Techniques: Introduction

$$\Pr[X \in \mathcal{R}] = \int_{\mathcal{R}} f(x) dx$$

- How can we approximate $\Pr[X \in \mathcal{R}_1]$ and $\Pr[X \in \mathcal{R}_2]$?
 - $\Pr[X \in \mathcal{R}_1] \approx \frac{6}{20}$ and $\Pr[X \in \mathcal{R}_2] \approx \frac{6}{20}$
- Should the density curves above \mathcal{R}_1 and \mathcal{R}_2 be equally high?
 - No, since \mathcal{R}_1 is smaller than \mathcal{R}_2

$$\Pr[X \in \mathcal{R}_1] = \int_{\mathcal{R}_1} f(x) dx \approx \int_{\mathcal{R}_2} f(x) dx = \Pr[X \in \mathcal{R}_2]$$

- To get density, normalize by region size



NonParametric Techniques: Introduction

- Assuming $f(x)$ is basically flat inside \mathcal{R} ,

$$\frac{\text{#of samples in } \mathcal{R}}{\text{total #of samples}} \approx \Pr[X \in \mathcal{R}] = \int_{\mathcal{R}} f(y) dy \approx f(x) * \text{Volume}(\mathcal{R})$$

- Thus, density at a point x inside \mathcal{R} can be approximated

$$f(x) \approx \frac{\text{#of samples in } \mathcal{R}}{\text{total #of samples}} \frac{1}{\text{Volume}(\mathcal{R})}$$

- Now let's derive this formula more formally

Binomial Random Variable

- Let us flip a coin n times (each one is called “trial”)
 - Probability of head ρ , probability of tail is $1-\rho$
- Binomial random variable K counts the number of heads in n trials

$$P(K = k) = \binom{n}{k} \rho^k (1 - \rho)^{n-k}$$

where $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

- Mean is $E(K) = n\rho$
- Variance is $\text{var}(K) = n\rho(1 - \rho)$

Density Estimation: Basic Issues

- From the definition of a density function, probability ρ that a vector \mathbf{x} will fall in region \mathcal{R} is:

$$\rho = \Pr[\mathbf{x} \in \mathcal{R}] = \int_{\mathcal{R}} p(\mathbf{x}') d\mathbf{x}'$$

- Suppose we have samples $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ drawn from the distribution $p(\mathbf{x})$. The probability that k points fall in \mathcal{R} is then given by binomial distribution:

$$\Pr[K = k] = \binom{n}{k} \rho^k (1 - \rho)^{n-k}$$

- Suppose that k points fall in \mathcal{R} , we can use MLE to estimate the value of ρ . The likelihood function is

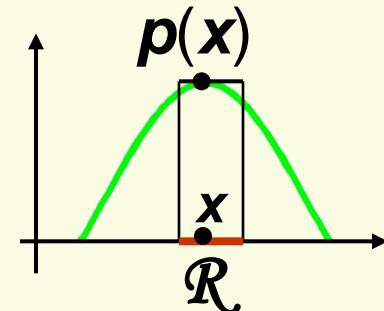
$$p(\mathbf{x}_1, \dots, \mathbf{x}_n | \rho) = \binom{n}{k} \rho^k (1 - \rho)^{n-k}$$

Density Estimation: Basic Issues

$$p(x_1, \dots, x_n | \rho) = \binom{n}{k} \rho^k (1 - \rho)^{n-k}$$

- This likelihood function is maximized at $\rho = \frac{k}{n}$
- Thus the MLE is $\hat{\rho} = \frac{k}{n}$
- Assume that $p(x)$ is continuous and that the region \mathcal{R} is so small that $p(x)$ is approximately constant in \mathcal{R}

$$\int_{\mathcal{R}} p(x') dx' \approx p(x) V$$

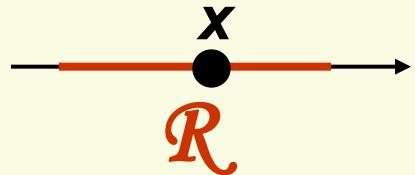


- x is in \mathcal{R} and V is the volume of \mathcal{R}
- Recall from the previous slide: $\rho = \int_{\mathcal{R}} p(x') dx'$
- Thus $p(x)$ can be approximated:
$$p(x) \approx \frac{k/n}{V}$$

Density Estimation: Basic Issues

- This is exactly what we had before:

$$p(x) \approx \frac{k/n}{V}$$



x is inside some region \mathcal{R}

k = number of samples inside \mathcal{R}

n = total number of samples

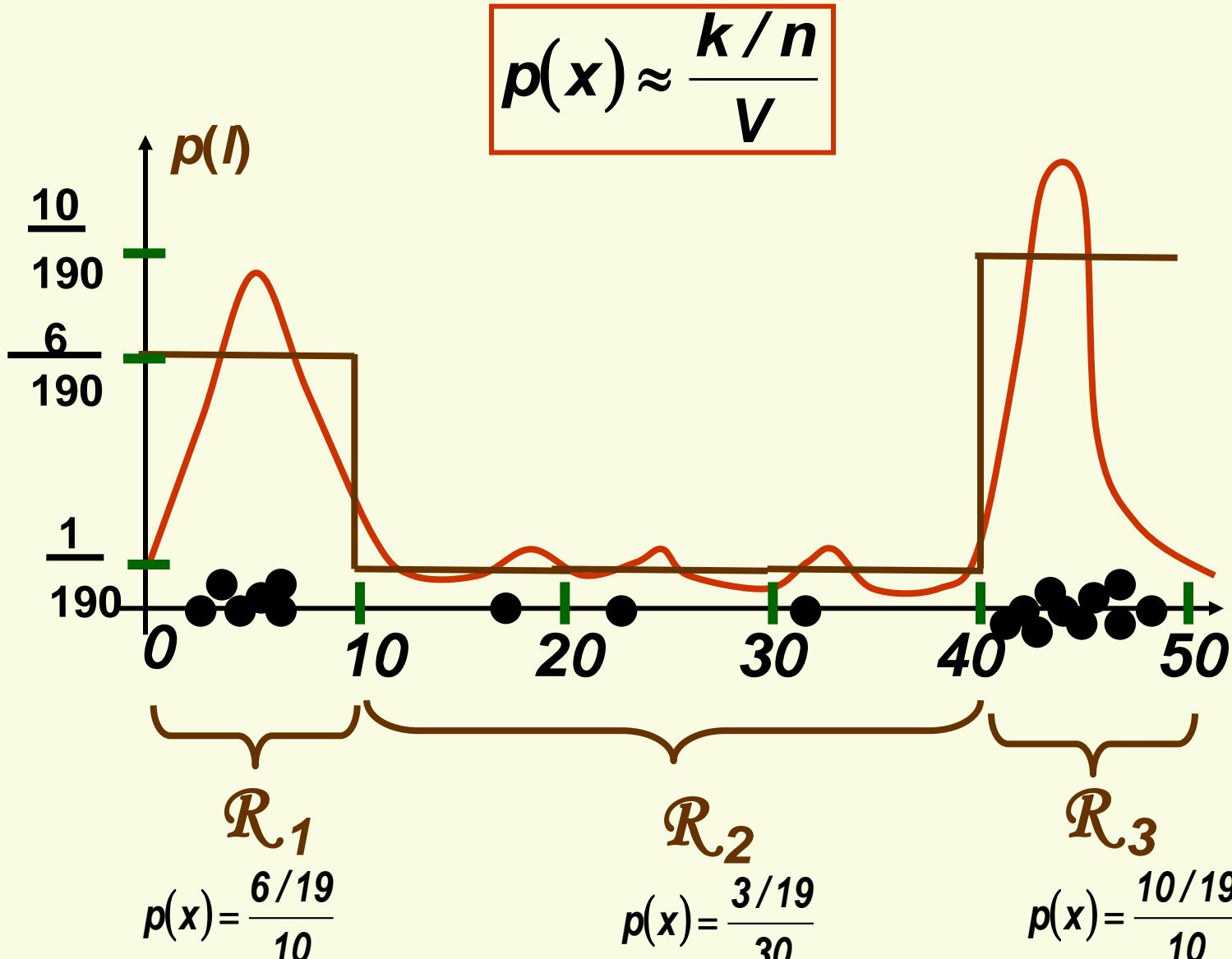
V = volume of \mathcal{R}

- Our estimate will always be the average of true density over \mathcal{R}

$$p(x) \approx \frac{k/n}{V} = \frac{\hat{\rho}}{V} \approx \frac{\int_{\mathcal{R}} p(x') dx'}{V}$$

- Ideally, $p(x)$ should be constant inside \mathcal{R}

Density Estimation: Histogram



- If regions \mathcal{R}_i 's do not overlap, we have a histogram

Density Estimation: Accuracy

- How accurate is density approximation $p(x) \approx \frac{k/n}{V}$?

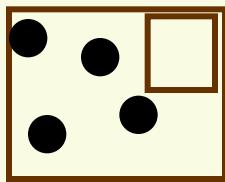
- We have made two approximations

1. $\hat{p} = \frac{k}{n}$

- as n increases, this estimate becomes more accurate

2. $\int_{\mathcal{R}} p(x') dx' \approx p(x)V$

- as \mathcal{R} grows smaller, the estimate becomes more accurate



- As we shrink \mathcal{R} we have to make sure it contains samples, otherwise our estimated $p(x) = 0$ for all x in \mathcal{R}

- Thus in theory, if we have an unlimited number of samples, we get convergence as we simultaneously increase the number of samples n , and shrink region \mathcal{R} , but not too much so that \mathcal{R} still contains a lot of samples

Density Estimation: Accuracy

$$p(x) \approx \frac{k/n}{V}$$

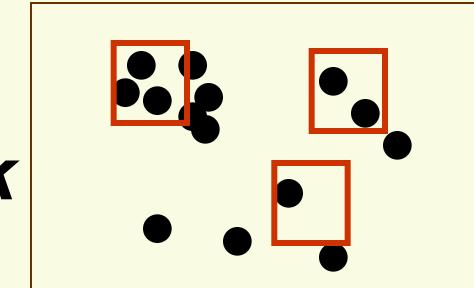
- In practice, the number of samples is always fixed
- Thus the only available option to increase the accuracy is by decreasing the size of \mathcal{R} (V gets smaller)
 - If V is too small, $p(x)=0$ for most x , because most regions will have no samples
 - Thus have to find a compromise for V
 - not too small so that it has enough samples
 - but also not too large so that $p(x)$ is approximately constant inside V

Density Estimation: Two Approaches

$$p(x) \approx \frac{k/n}{V}$$

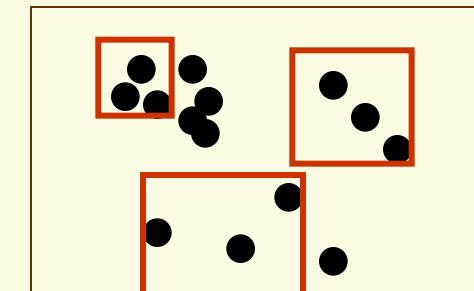
1. Parzen Windows:

- Choose a fixed value for volume V and determine the corresponding k from the data



2. k-Nearest Neighbors

- Choose a fixed value for k and determine the corresponding volume V from the data
- Under appropriate conditions and as number of samples goes to infinity, both methods can be shown to converge to the true $p(x)$



Parzen Windows

$$p(x) \approx \frac{k/n}{V}$$

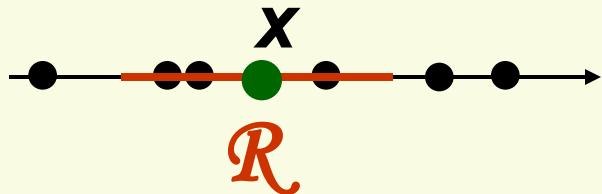
x is inside some region \mathcal{R}

k = number of samples inside \mathcal{R}

n=total number of samples

V = volume of \mathcal{R}

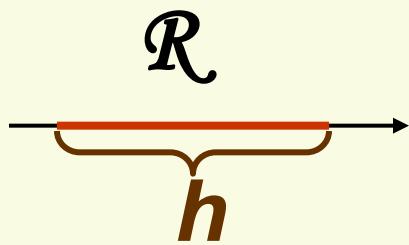
- To estimate the density at point x , simply center the region \mathcal{R} at x , count the number of samples in \mathcal{R} , and substitute everything in our formula



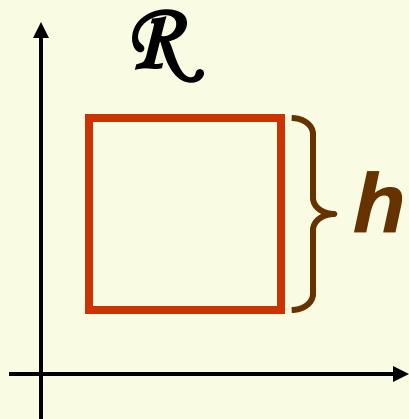
$$p(x) \approx \frac{3/6}{10}$$

Parzen Windows

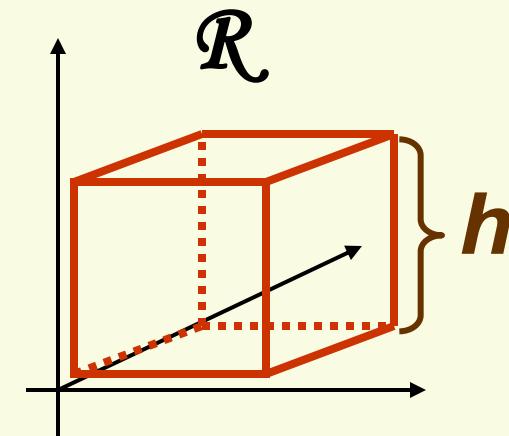
- In Parzen-window approach to estimate densities we fix the size and shape of region \mathcal{R}
- Let us assume that the region \mathcal{R} is a d -dimensional hypercube with side length h thus it's volume is h^d



1 dimension



2 dimensions

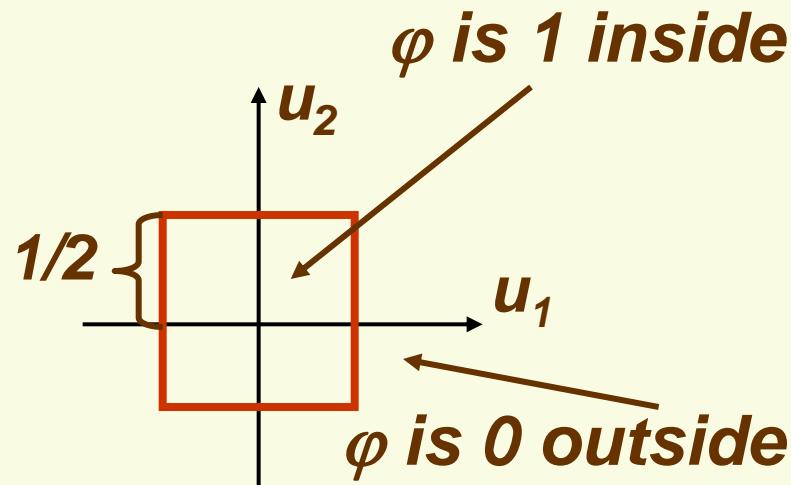
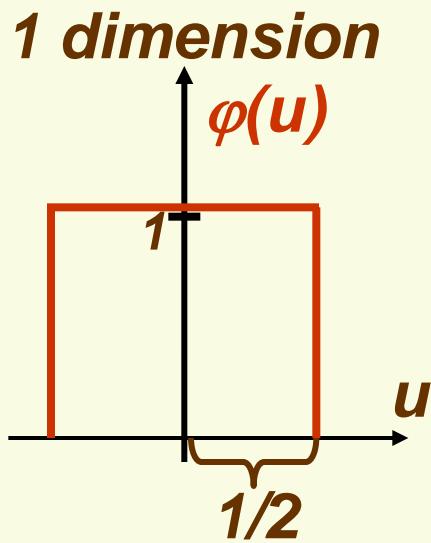


3 dimensions

Parzen Windows

- Let $u = [u_1, u_2, \dots, u_d]$ and define a window function

$$\varphi(u) = \begin{cases} 1 & |u_j| \leq \frac{1}{2} \quad j = 1, \dots, d \\ 0 & \text{otherwise} \end{cases}$$



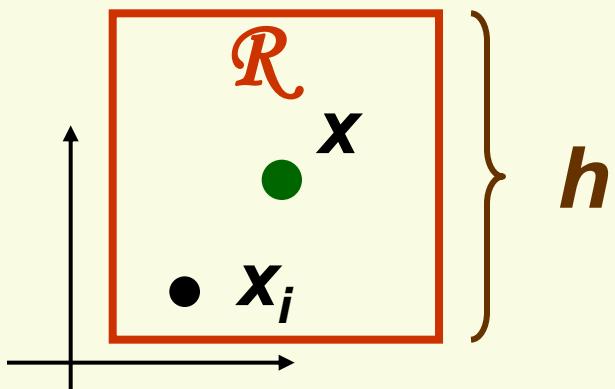
2 dimensions

Parzen Windows

- Recall we have d-dimensional samples x_1, x_2, \dots, x_n . Let x_{ij} be the jth coordinate of sample x_i . Then

$$\varphi\left(\frac{x - x_i}{h}\right) = \begin{cases} 1 & |x_j - x_{ij}| \leq \frac{h}{2}, \quad j = 1, \dots, d \\ 0 & \text{otherwise} \end{cases}$$

$|u_j| \leq \frac{1}{2}$



The diagram illustrates a 2D coordinate system with a horizontal and vertical axis. A red square, labeled \mathcal{R} , represents a hypercube of width h centered at a point x_i marked with a black dot. Inside the square, another point x is marked with a green dot, indicating it is within the window. The distance from x to the nearest edge of the square is labeled u_j .

$$\varphi\left(\frac{x - x_i}{h}\right) = \begin{cases} 1 & \text{if } x_i \text{ is inside the hypercube with} \\ & \text{width } h \text{ and centered at } x \\ 0 & \text{otherwise} \end{cases}$$

Parzen Windows

- How do we count the total number of sample points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ which are inside the hypercube with side h and centered at \mathbf{x} ?

$$k = \sum_{i=1}^{i=n} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

- Recall $p(\mathbf{x}) \approx \frac{k/n}{V}$, $V=h^d$
- Thus we get the desired analytical expression for the estimate of density $p_\varphi(\mathbf{x})$

$$p_\varphi(\mathbf{x}) = \frac{\sum_{i=1}^{i=n} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) / n}{h^d} = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

Parzen Windows

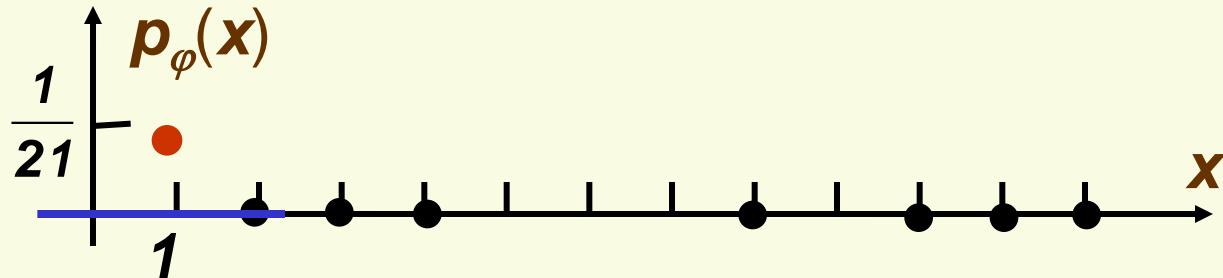
$$p_\varphi(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)$$

- Let's make sure $p_\varphi(\mathbf{x})$ is in fact a density
- $p_\varphi(\mathbf{x}) \geq 0 \quad \forall \mathbf{x}$
- $\int p_\varphi(\mathbf{x}) d\mathbf{x} = \int \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) d\mathbf{x} = \frac{1}{h^d n} \sum_{i=1}^{i=n} \underbrace{\int \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) d\mathbf{x}}_{\text{volume of hypercube}}$
$$= \frac{1}{n} \frac{1}{h^d} \sum_{i=1}^{i=n} h^d = 1$$

Parzen Windows: Example in 1D

$$p_\phi(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{x - x_i}{h}\right)$$

- Suppose we have 7 samples $D=\{2,3,4,8,10,11,12\}$



- Let window width $h=3$, estimate density at $x=1$

$$p_\phi(1) = \frac{1}{7} \sum_{i=1}^{i=7} \frac{1}{3} \varphi\left(\frac{1-x_i}{3}\right) = \frac{1}{21} \left[\varphi\left(\frac{1-2}{3}\right) + \varphi\left(\frac{1-3}{3}\right) + \varphi\left(\frac{1-4}{3}\right) + \dots + \varphi\left(\frac{1-12}{3}\right) \right]$$
$$\left|-\frac{1}{3}\right| \leq 1/2 \quad \left|-\frac{2}{3}\right| > 1/2 \quad \left|-1\right| > 1/2 \quad \left|-\frac{11}{3}\right| > 1/2$$

$$p_\phi(1) = \frac{1}{7} \sum_{i=1}^{i=7} \frac{1}{3} \varphi\left(\frac{1-x_i}{3}\right) = \frac{1}{21} [1 + 0 + 0 + \dots + 0] = \frac{1}{21}$$

Parzen Windows: Sum of Functions

- Now let's look at our density estimate $p_\phi(\mathbf{x})$ again:

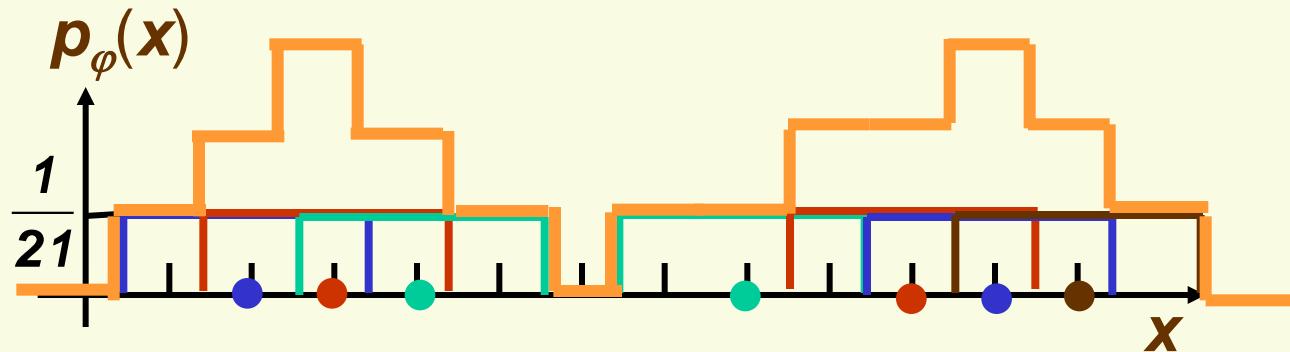
$$p_\phi(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) = \sum_{i=1}^{i=n} \underbrace{\frac{1}{nh^d} \varphi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right)}$$

**1 inside square centered at \mathbf{x}_i ,
0 otherwise**

- Thus $p_\phi(\mathbf{x})$ is just a sum of n “box like” functions each of height $\frac{1}{nh^d}$

Parzen Windows: Example in 1D

- Let's come back to our example
 - 7 samples $D=\{2,3,4,8,10,11,12\}$, $h=3$

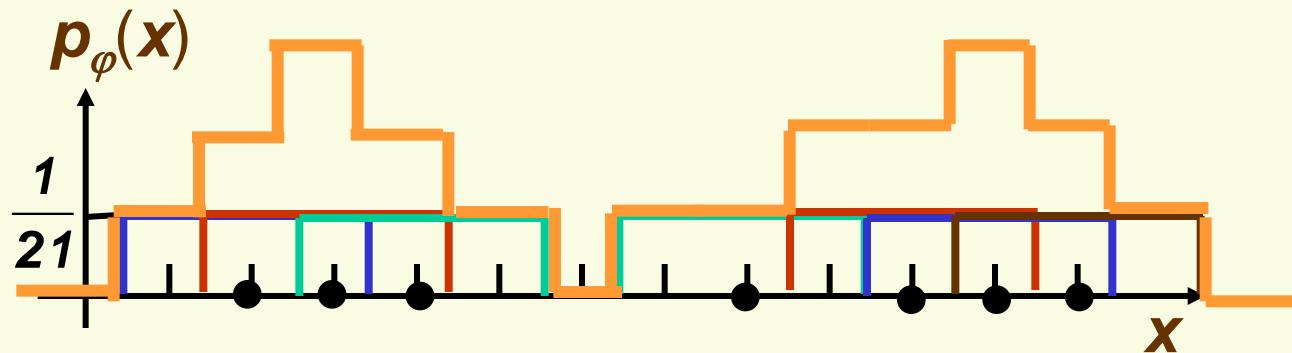


- To see what the function looks like, we need to generate 7 boxes and add them up
- The width is $h=3$ and the height is

$$\frac{1}{nh^d} = \frac{1}{21}$$

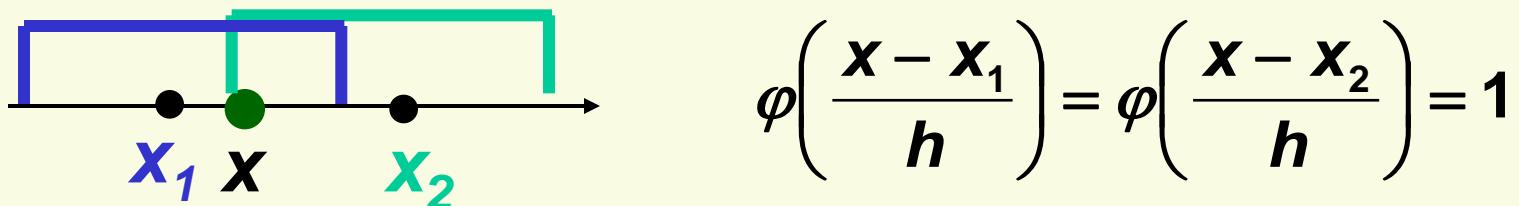
Parzen Windows: Interpolation

- In essence, window function φ is used for interpolation: each sample x_i contributes to the resulting density at x if x is close enough to x_i

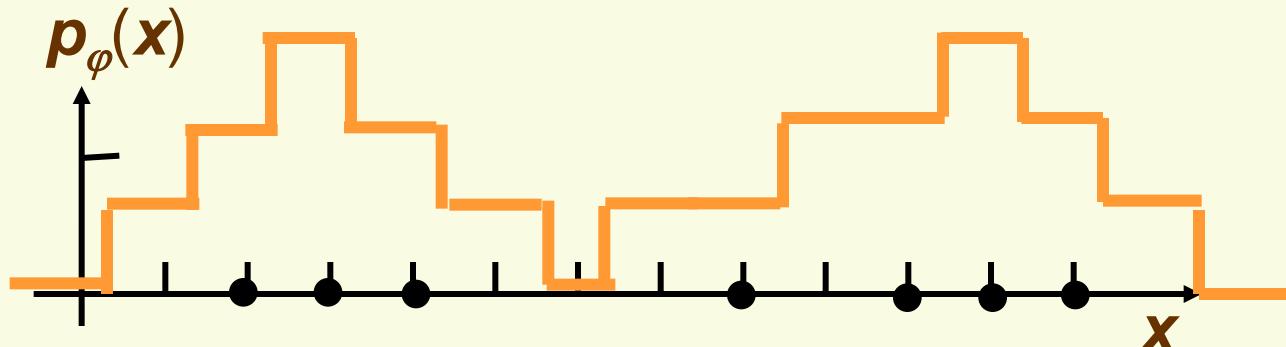


Parzen Windows: Drawbacks of Hypercube φ

- As long as sample point x_i and x are in the same hypercube, the contribution of x_i to the density at x is constant, regardless of how close x_i is to x



- The resulting density $p_\varphi(x)$ is not smooth, it has discontinuities



Parzen Windows: general φ

$$p_{\varphi}(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{x - x_i}{h}\right)$$

- We can use a general window φ as long as the resulting $p_{\varphi}(x)$ is a legitimate density, i.e.

1. $p_{\varphi}(u) \geq 0$

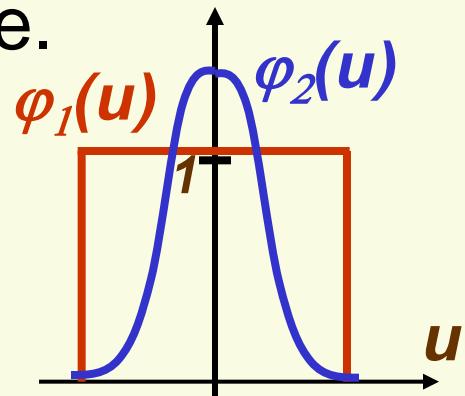
- satisfied if $\varphi(u) \geq 0$

2. $\int p_{\varphi}(x) dx = 1$

- satisfied if $\int \varphi(u) du = 1$

$$\int p_{\varphi}(x) dx = \frac{1}{nh^d} \sum_{i=1}^{i=n} \int \varphi\left(\frac{x - x_i}{h}\right) dx = \frac{1}{nh^d} \sum_{i=1}^n \int h^d \varphi\left(\frac{x - x_i}{h}\right) du = 1$$

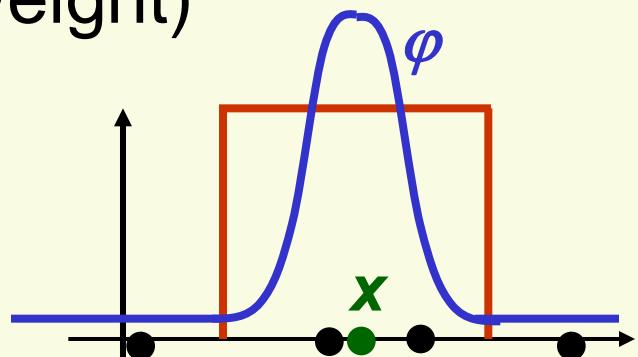
change coordinates to $u = \frac{x - x_i}{h}$, thus $du = \frac{dx}{h}$



Parzen Windows: general φ

$$p_{\varphi}(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{x - x_i}{h}\right)$$

- Notice that with the general window φ we are no longer counting the number of samples inside \mathcal{R} .
- We are counting the weighted average of potentially every single sample point (although only those within distance h have any significant weight)



- With infinite number of samples, and appropriate conditions, it can still be shown that

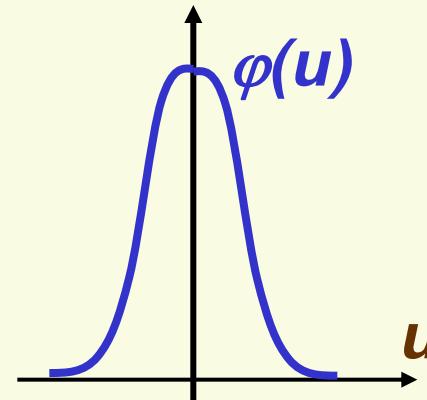
$$p_{\varphi}^n(x) \rightarrow p(x)$$

Parzen Windows: Gaussian φ

$$p_{\varphi}(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{x - x_i}{h}\right)$$

- A popular choice for φ is $N(0, 1)$ density

$$\varphi(u) = \frac{1}{\sqrt{2\pi}} e^{-u^2/2}$$

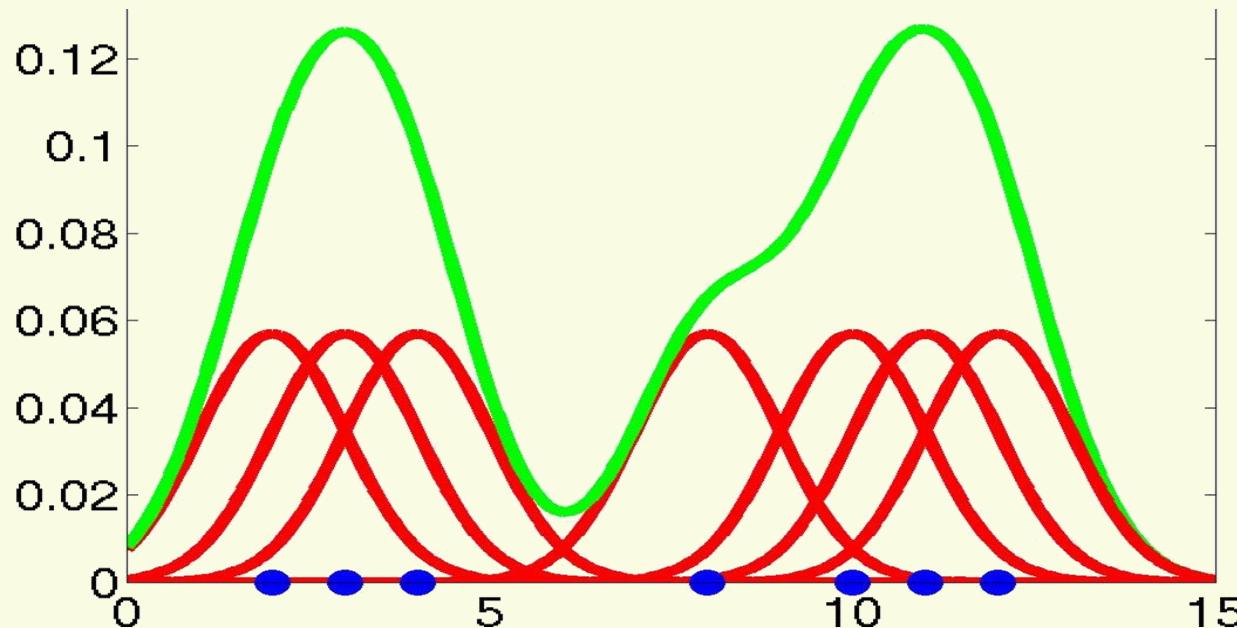


- Solves both drawbacks of the “box” window
 - Points x which are close to the sample point x_i receive higher weight
 - Resulting density $p_{\varphi}(x)$ is smooth

Parzen Windows: Example with General φ

- Let's come back to our example
 - 7 samples $D=\{2,3,4,8,10,11,12\}$, $h=1$

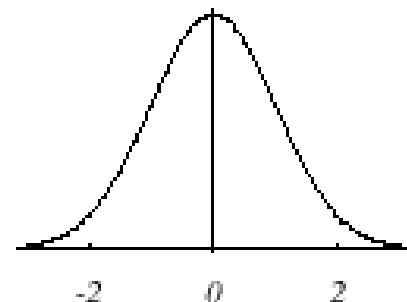
$$p_{\varphi}(x) = \frac{1}{7} \sum_{i=1}^{i=7} \varphi(x - x_i)$$



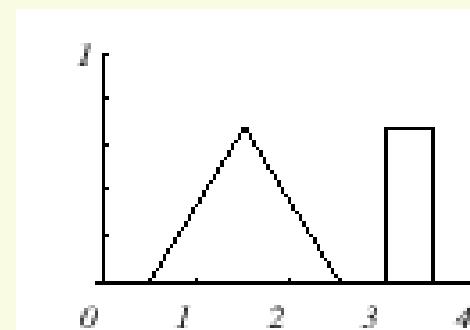
- $p_{\varphi}(x)$ is the sum of 7 Gaussians, each centered at one of the sample points, and each scaled by $1/7$

Parzen Windows: Did We Solve the Problem?

- Let's test if we solved the problem
 - Draw samples from a known distribution
 - Use our density approximation method and compare with the true density
- We will vary the number of samples n and the window size h
- We will play with 2 distributions

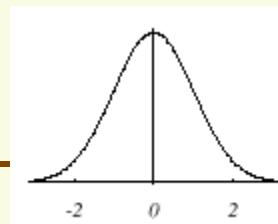


$N(0, 1)$

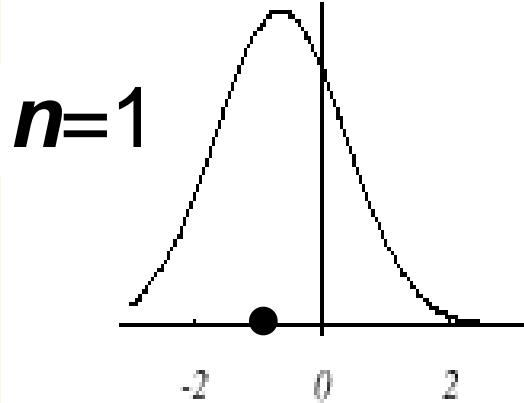


*triangle and
uniform mixture*

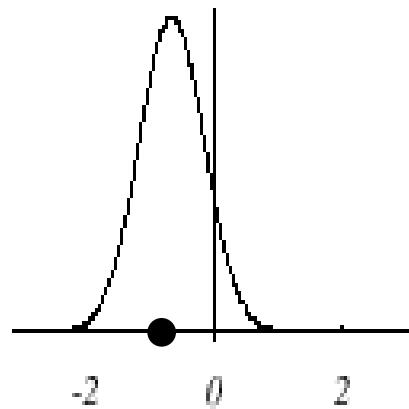
Parzen Windows: True Density $N(0, 1)$



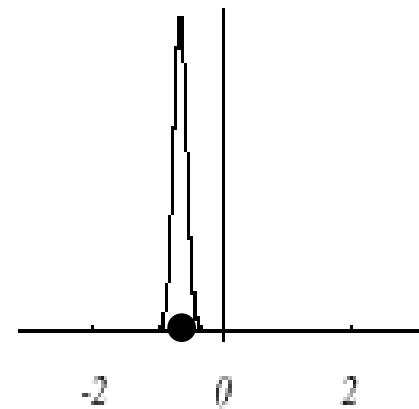
$h=1$



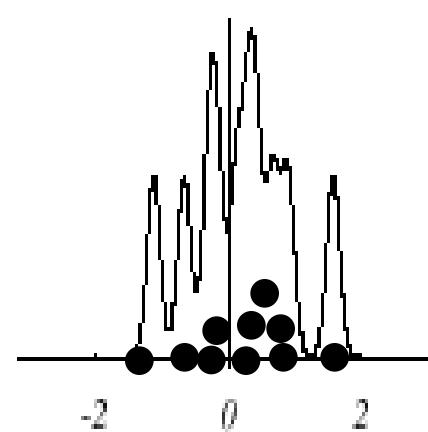
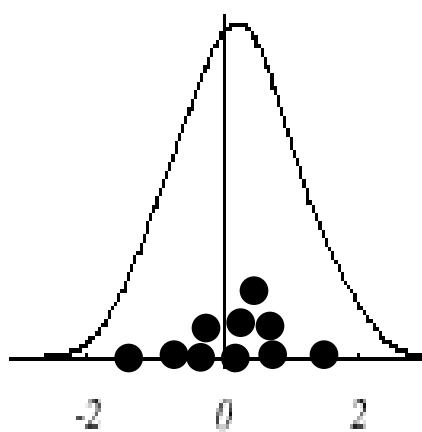
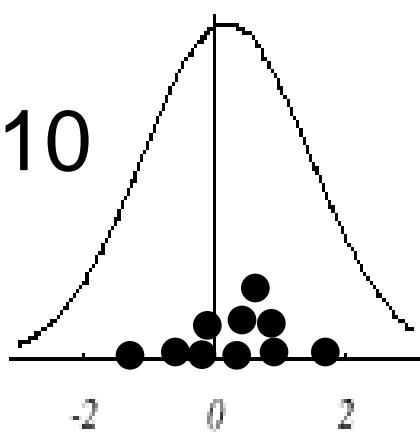
$h=0.5$



$h=0.1$



$n=10$



Parzen Windows: True Density $N(0, 1)$

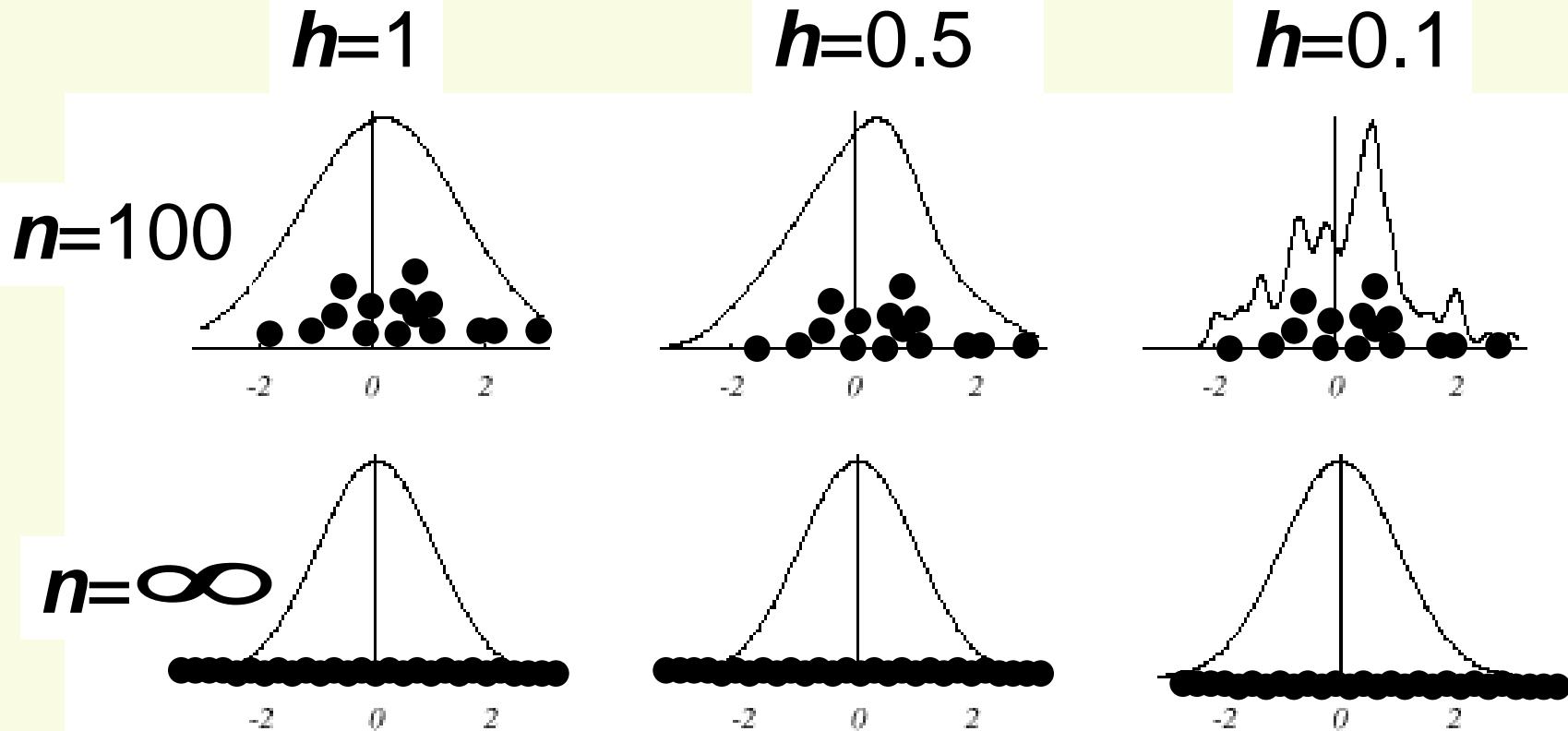
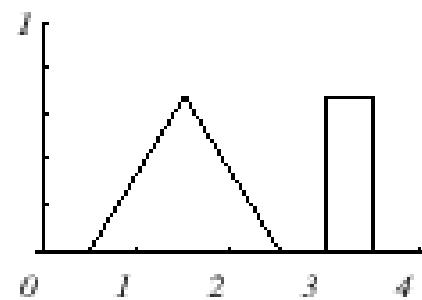


FIGURE 4.5. Parzen-window estimates of a univariate normal density using different window widths and numbers of samples. The vertical axes have been scaled to best show the structure in each graph. Note particularly that the $n = \infty$ estimates are the same (and match the true density function), regardless of window width. From: Richard

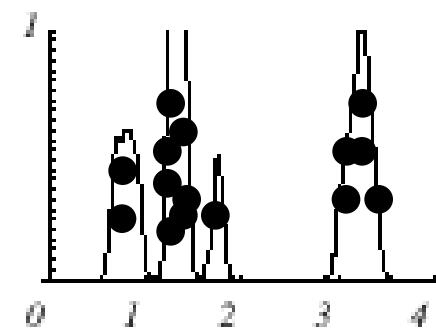
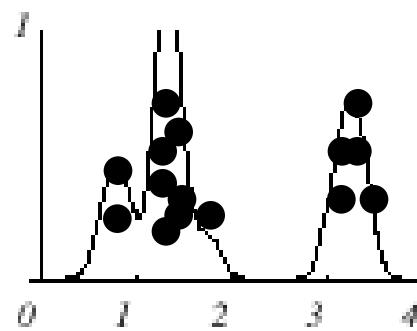
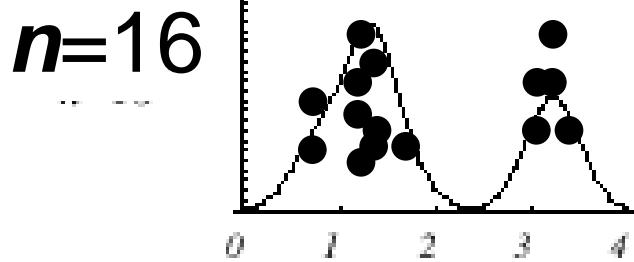
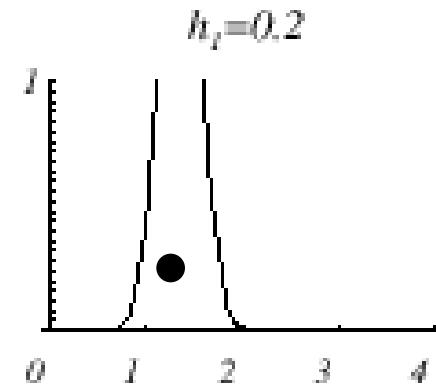
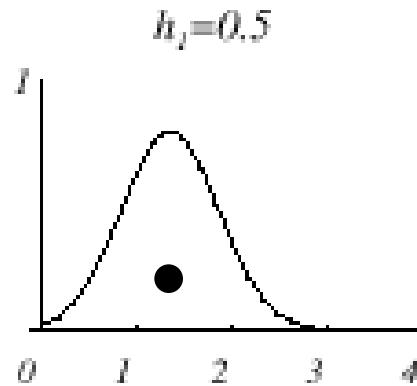
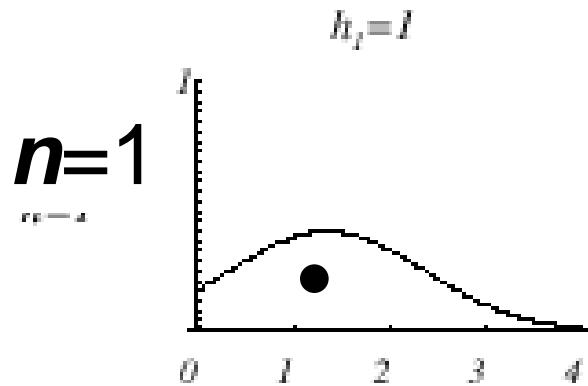
Parzen Windows: True density is Mixture of Uniform and Triangle



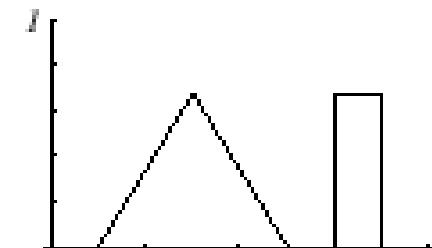
$h=1$

$h=0.5$

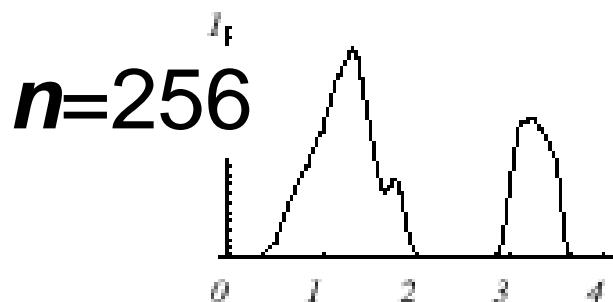
$h=0.2$



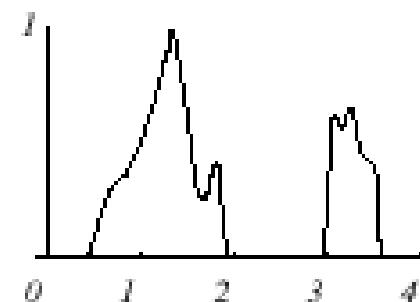
Parzen Windows: True density is Mixture of Uniform and Triangle



$h=1$



$h=0.5$



$h=0.2$

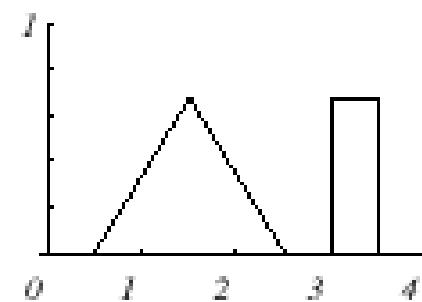
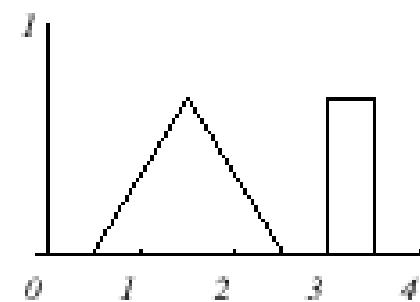
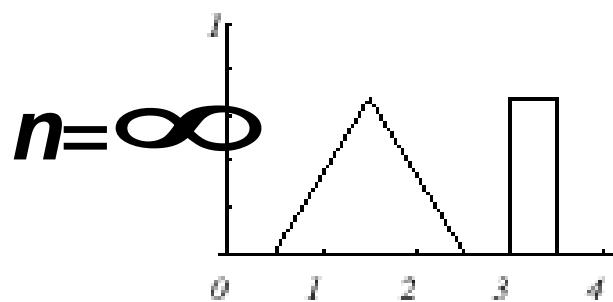
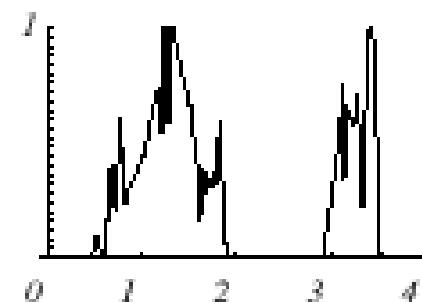
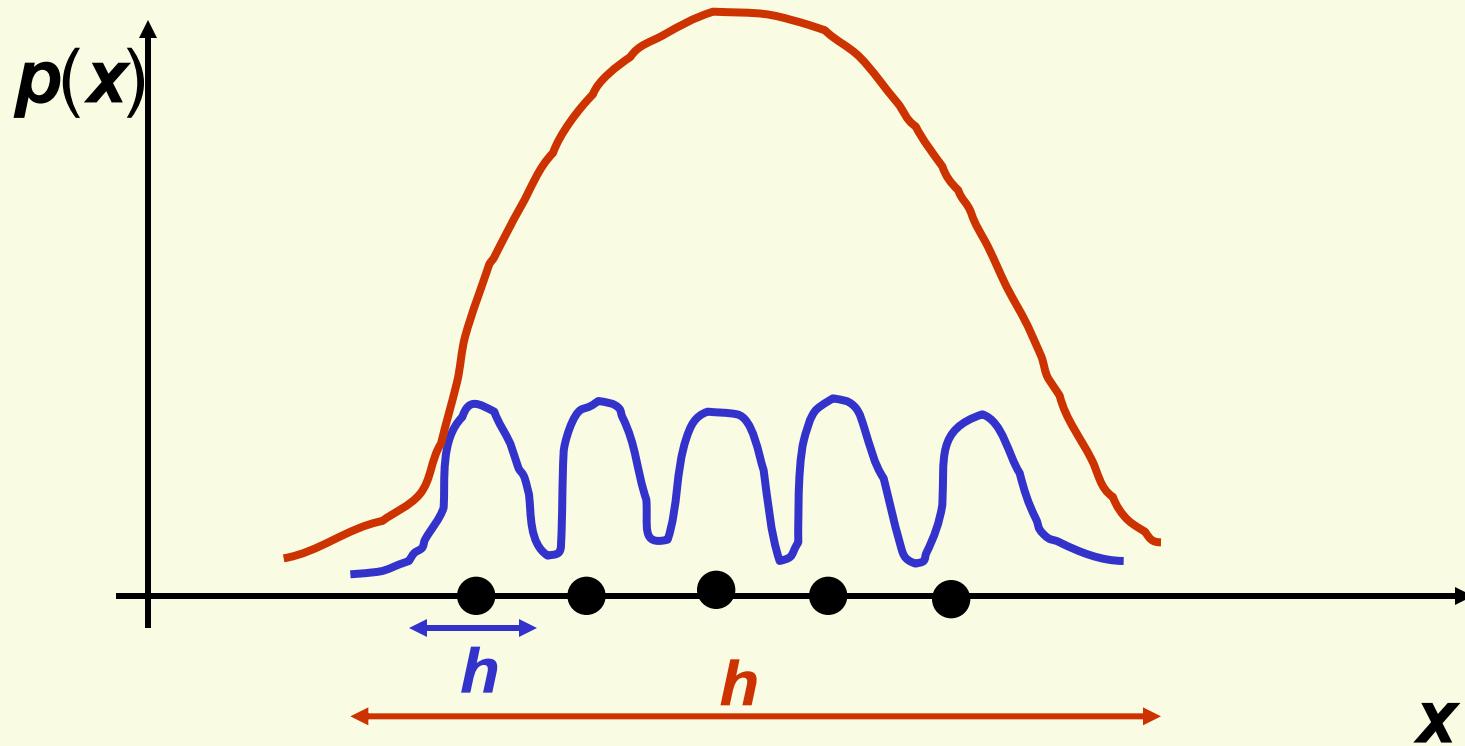


FIGURE 4.7. Parzen-window estimates of a bimodal distribution using different window widths and numbers of samples. Note particularly that the $n = \infty$ estimates are the same (and match the true distribution), regardless of window width. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Parzen Windows: Effect of Window Width h

- By choosing h we are guessing the region where density is approximately constant
- Without knowing anything about the distribution, it is really hard to guess where the density is approximately constant



Parzen Windows: Effect of Window Width h

- If h is small, we superimpose n sharp pulses centered at the data
 - Each sample point x_i , influences too small range of x
 - Smoothed too little: the result will look noisy and not smooth enough
- If h is large, we superimpose broad slowly changing functions,
 - Each sample point x_i , influences too large range of x
 - Smoothed too much: the result looks oversmoothed or “out-of-focus”
- Finding the best h is challenging, and indeed no single h may work well
 - May need to adapt h for different sample points
- However we can try to learn the best h to use from our labeled data

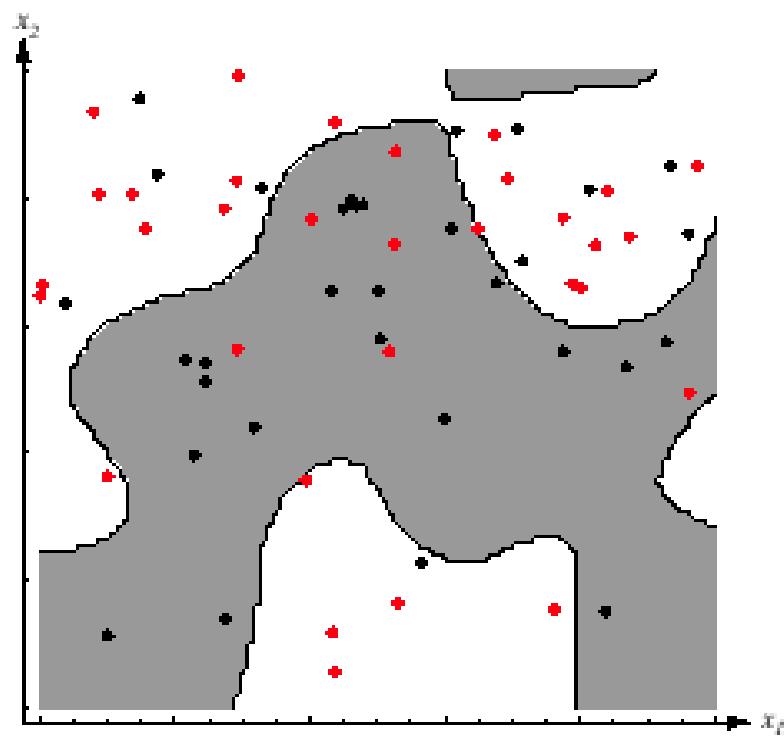
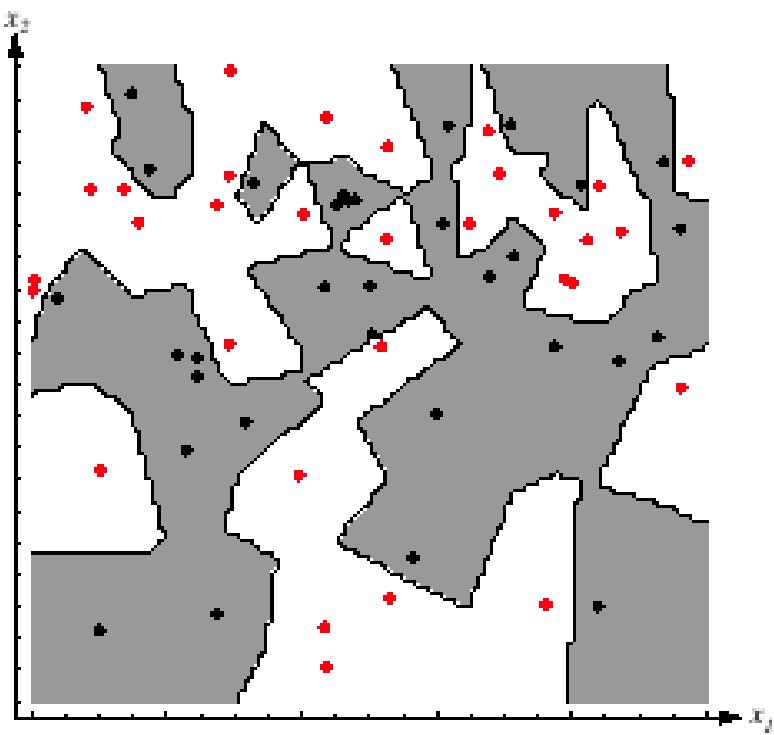
Learning window width h From Labeled Data

- Divide labeled data into *training* set, *validation* set, *test* set
- For a range of different values of h (possibly using binary search), construct density estimate $p(x)$ using Parzen windows
- Test the classification performance on the ***validation*** set for each value of h you tried
- For the final density estimate, choose h giving the smallest error on the ***validation*** set
- Now you can test the performance of the classifier on the test set
 - Notice we need validation set to find best parameter h , we can't use test set for this because test set cannot be used for training
 - In general, need validation set if our classifier has some tunable parameters

Parzen Windows: Classification Example

- In classifiers based on Parzen-window estimation:
 - We estimate the densities for each category and classify a test point by the label corresponding to the maximum posterior
 - The decision region for a Parzen-window classifier depends upon the choice of window function as illustrated in the following figure

Parzen Windows: Classification Example



- For **small** enough window size h the classification on training data is perfect
- However decision boundaries are complex and this solution is not likely to generalize well to novel data
- For **larger** window size h , classification on training data is not perfect
- However decision boundaries are simpler and this solution is more likely to generalize well to novel data

Parzen Windows: Summary

- Advantages
 - Can be applied to the data from any distribution
 - In theory can be shown to converge as the number of samples goes to infinity
- Disadvantages
 - Number of training data is limited in practice, and so choosing the appropriate window size h is difficult
 - May need large number of samples for accurate estimates
 - Computationally heavy, to classify one point we have to compute a function which potentially depends on all samples

$$p_{\varphi}(x) = \frac{1}{n} \sum_{i=1}^{i=n} \frac{1}{h^d} \varphi\left(\frac{x - x_i}{h}\right)$$

- **But we need a lot of samples for accurate density estimation!**

Nonparametric Density Estimation
Nearest Neighbors , KNN

k-Nearest Neighbors

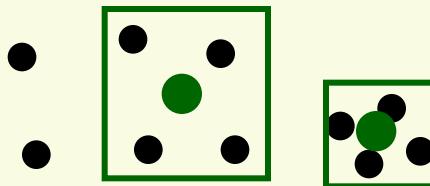
- Recall the generic expression for density estimation

$$p(x) \approx \frac{k/n}{V}$$

- In Parzen windows estimation, we fix V and that determines k , the number of points inside V
- In k-nearest neighbor approach we fix k , and find V that contains k points inside

***k*-Nearest Neighbors**

- kNN approach seems a good solution for the problem of the “best” window size
 - Let the cell volume be a function of the training data
 - Center a cell about x and let it grows until it captures k samples
 - k are called the k nearest-neighbors of x

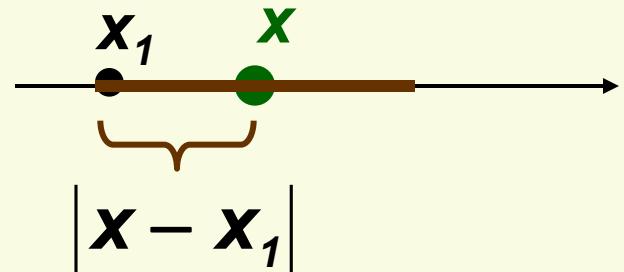


- 2 possibilities can occur:
 - Density is high near x ; therefore the cell will be small which provides a good resolution
 - Density is low; therefore the cell will grow large and stop until higher density regions are reached

k-Nearest Neighbor

- Of course, now we have a new question
 - How to choose k ?
- A good “rule of thumb“ is $k = \sqrt{n}$
 - Can prove convergence if n goes to infinity
 - Not too useful in practice, however
- Let’s look at 1-D example
 - we have one sample, i.e. $n = 1$

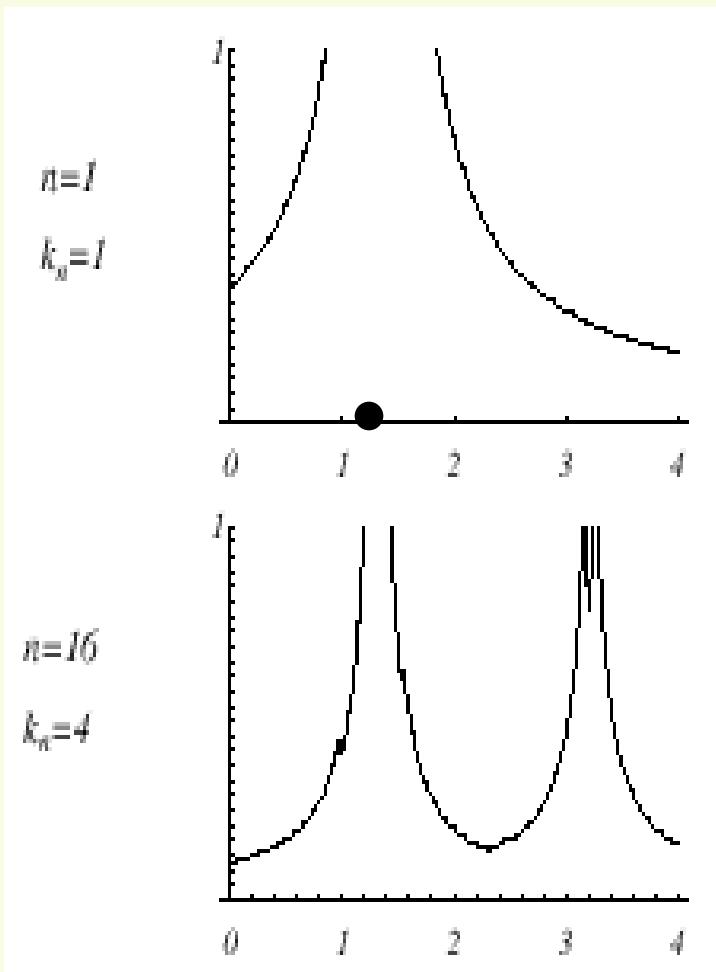
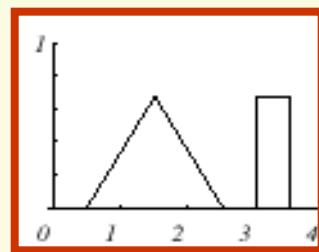
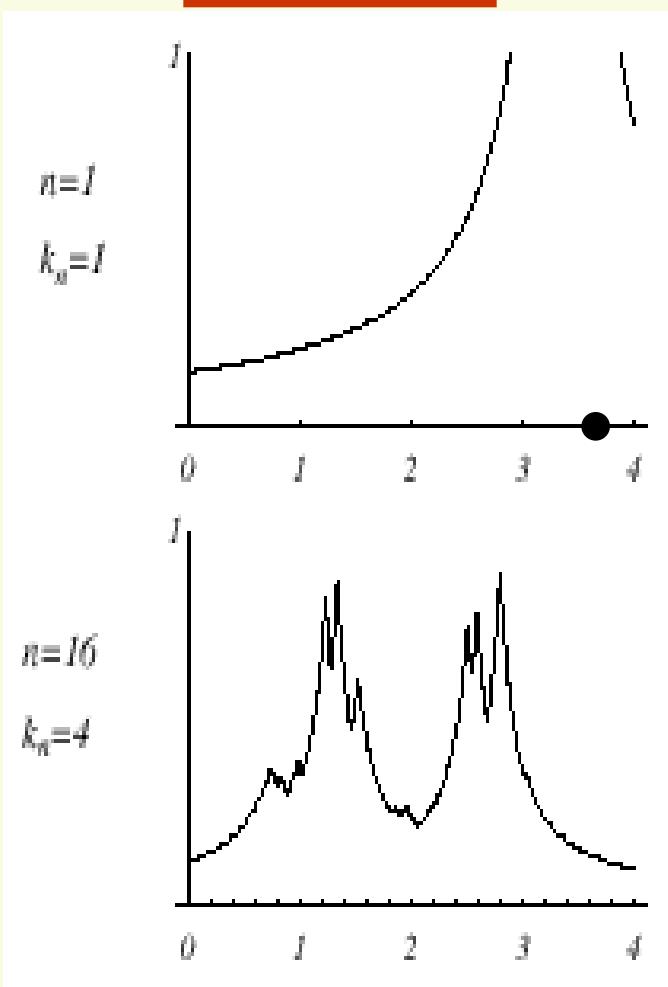
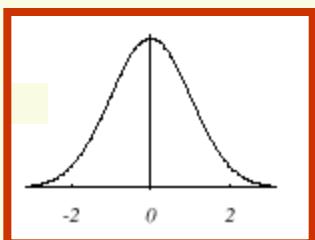
$$p(x) \approx \frac{k/n}{V} = \frac{1}{2|x - x_1|}$$



- But the estimated $p(x)$ is not even close to a density function:

$$\int_{-\infty}^{\infty} \frac{1}{2|x - x_1|} dx = \infty \neq 1$$

k-Nearest Neighbor: Density estimation



k-Nearest Neighbor

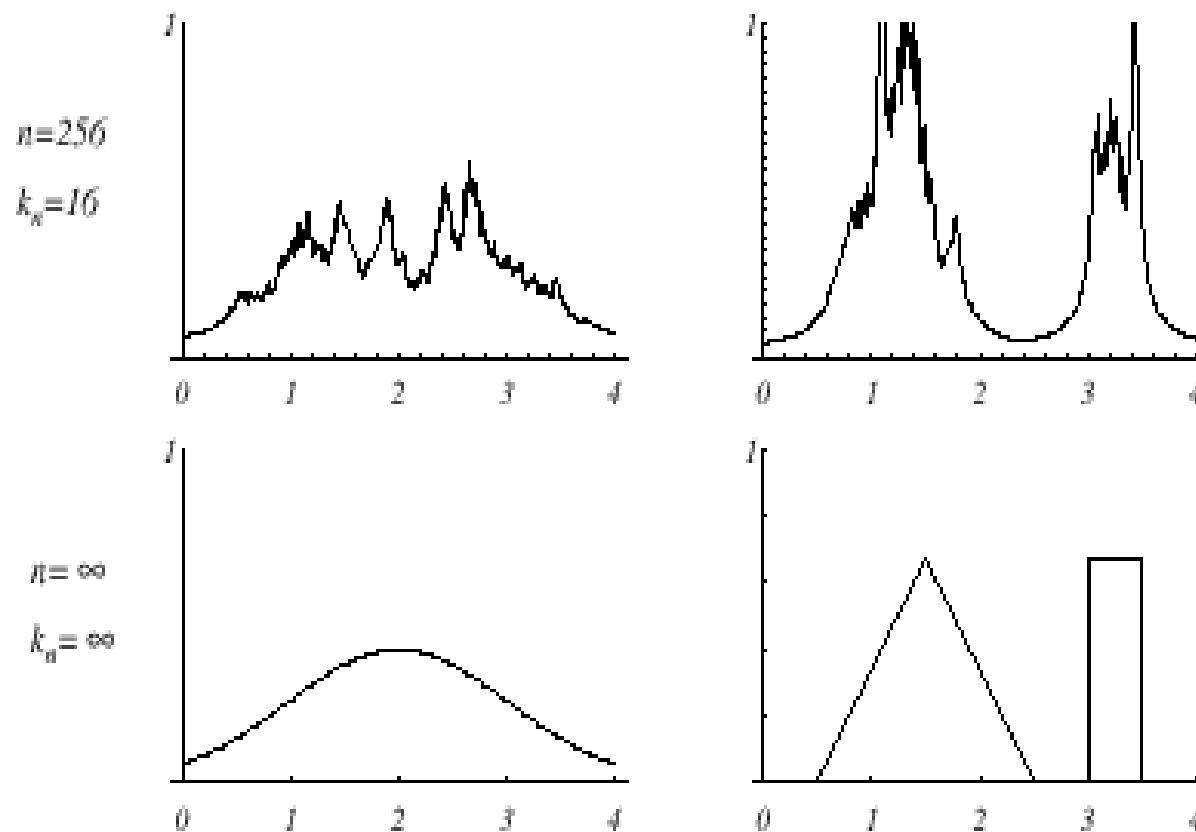


FIGURE 4.12. Several k -nearest-neighbor estimates of two unidimensional densities: a Gaussian and a bimodal distribution. Notice how the finite n estimates can be quite "spiky." From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

k-Nearest Neighbor

- Thus straightforward density estimation $p(\mathbf{x})$ does not work very well with kNN approach because the resulting density estimate
 1. Is not even a density
 2. Has a lot of discontinuities (looks very spiky, not differentiable)
 3. Even for large regions with no observed samples the estimated density is far from zero (tails are too heavy)
- *Notice in the theory, if infinite number of samples is available, we could construct a series of estimates that converge to the true density using kNN estimation. However this theorem is not very useful in practice because the number of samples is always limited*

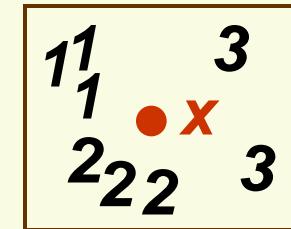
k-Nearest Neighbor

- However we shouldn't give up the nearest neighbor approach yet
- Instead of approximating the density $p(\mathbf{x})$, we can use kNN method to approximate the posterior distribution $P(c_i|\mathbf{x})$
 - We don't need $p(\mathbf{x})$ if we can get a good estimate on $P(c_i|\mathbf{x})$

***k*-Nearest Neighbor**

- How would we estimate $P(c_i | x)$ from a set of n labeled samples?
- Recall our estimate for density: $p(x) \approx \frac{k/n}{V}$
- Let's place a cell of volume V around x and capture k samples
 - k_i samples amongst k labeled c_i , then:

$$p(c_i, x) \approx \frac{k_i / n}{V}$$



- Using conditional probability, let's estimate posterior:

$$p(c_i | x) = \frac{p(x, c_i)}{p(x)} = \frac{p(x, c_i)}{\sum_{j=1}^m p(x, c_j)} \approx \frac{k_i / n}{V \sum_{j=1}^m k_j / n} = \frac{k_i}{\sum_{j=1}^m k_j} = \frac{k_i}{k}$$

k-Nearest Neighbor Rule

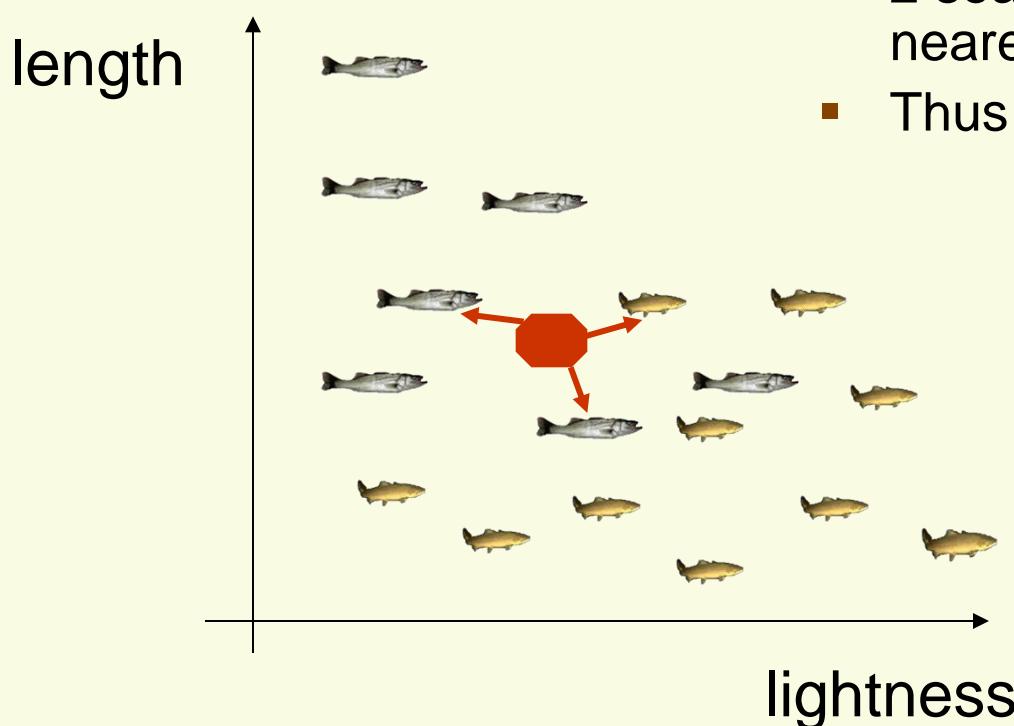
- Thus our estimate of posterior is just the fraction of samples which belong to class c_i :

$$p(c_i | x) = \frac{k_i}{k}$$

- This is a very simple and intuitive estimate
- Under the zero-one loss function (MAP classifier) just choose the class which has the largest number of samples in the cell
- Interpretation is: given an unlabeled example (that is x), find k most similar labeled examples (closest neighbors among sample points) and assign the most frequent class among those neighbors to x

k-Nearest Neighbor: Example

- Back to fish sorting
 - Suppose we have 2 features, and collected sample points as in the picture
 - Let $k = 3$



- 2 sea bass, 1 salmon are the 3 nearest neighbors
- Thus classify as sea bass

kNN: How Well Does it Work?

- kNN rule is certainly simple and intuitive, but does it work?
- Assume we have an unlimited number of samples
- By definition, the best possible error rate is the Bayes rate E^*
- Nearest-neighbor rule leads to an error rate greater than E^*
- But even for $k=1$, as $n \rightarrow \infty$, it can be shown that nearest neighbor rule error rate is smaller than $2E^*$
- As we increase k , the upper bound on the error gets better and better, that is the error rate (as $n \rightarrow \infty$) for the ***kNN*** rule is smaller than cE^* , with smaller c for larger k
- If we have a lot of samples, the kNN rule will do very well !

1NN: Voronoi Cells

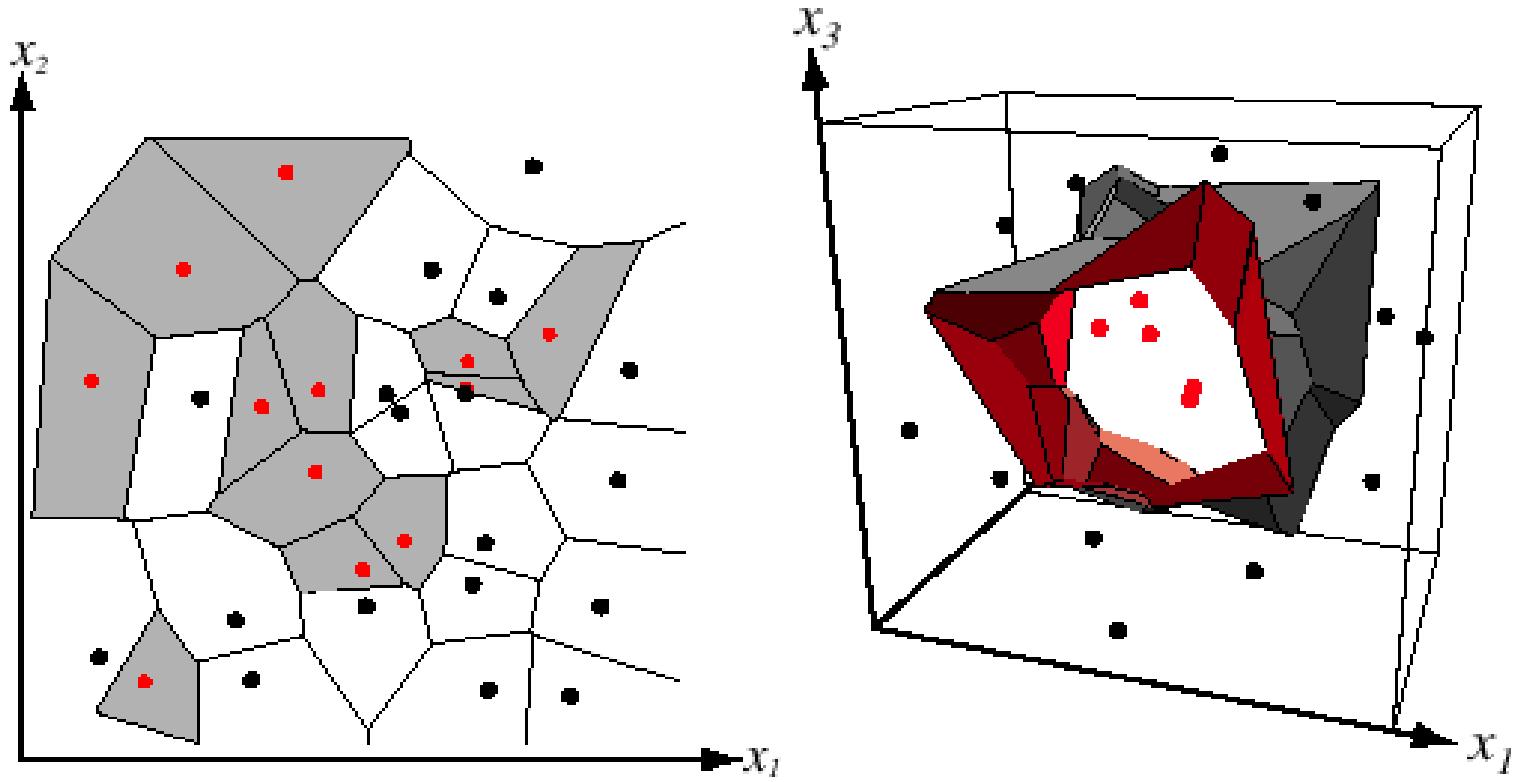
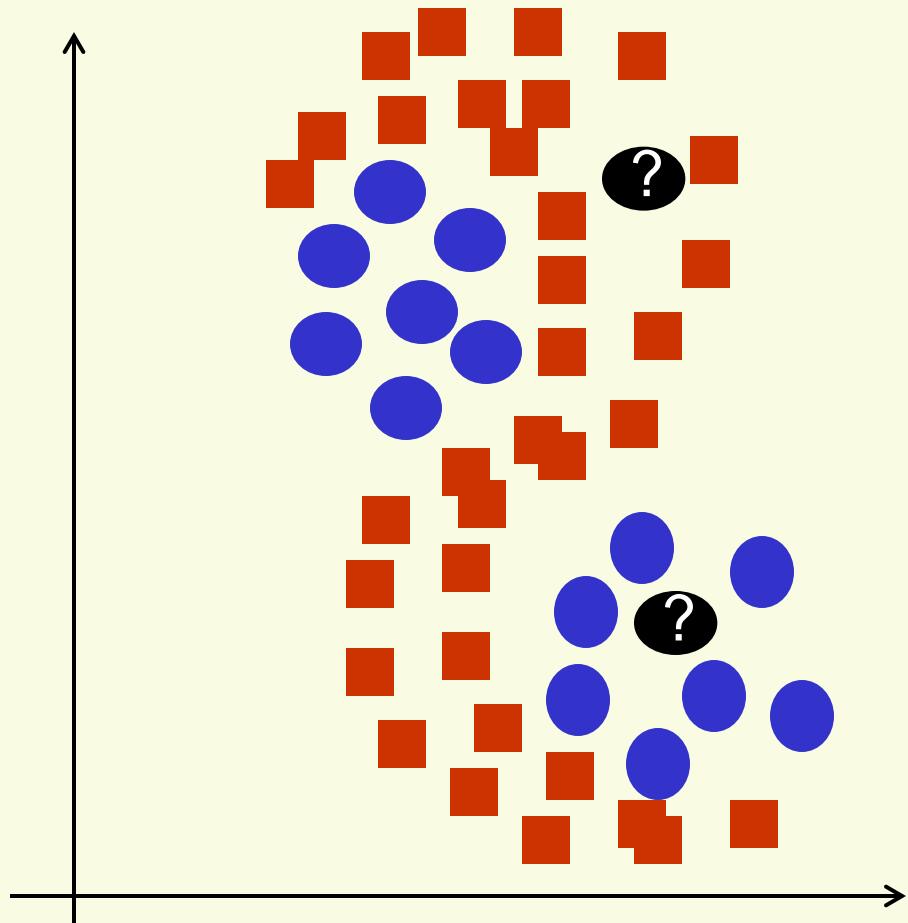


FIGURE 4.13. In two dimensions, the nearest-neighbor algorithm leads to a partitioning of the input space into Voronoi cells, each labeled by the category of the training point it contains. In three dimensions, the cells are three-dimensional, and the decision boundary resembles the surface of a crystal. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

kNN: Multi-Modal Distributions

- Most parametric distributions would not work for this 2 class classification problem:
- Nearest neighbors will do reasonably well, provided we have a lot of samples



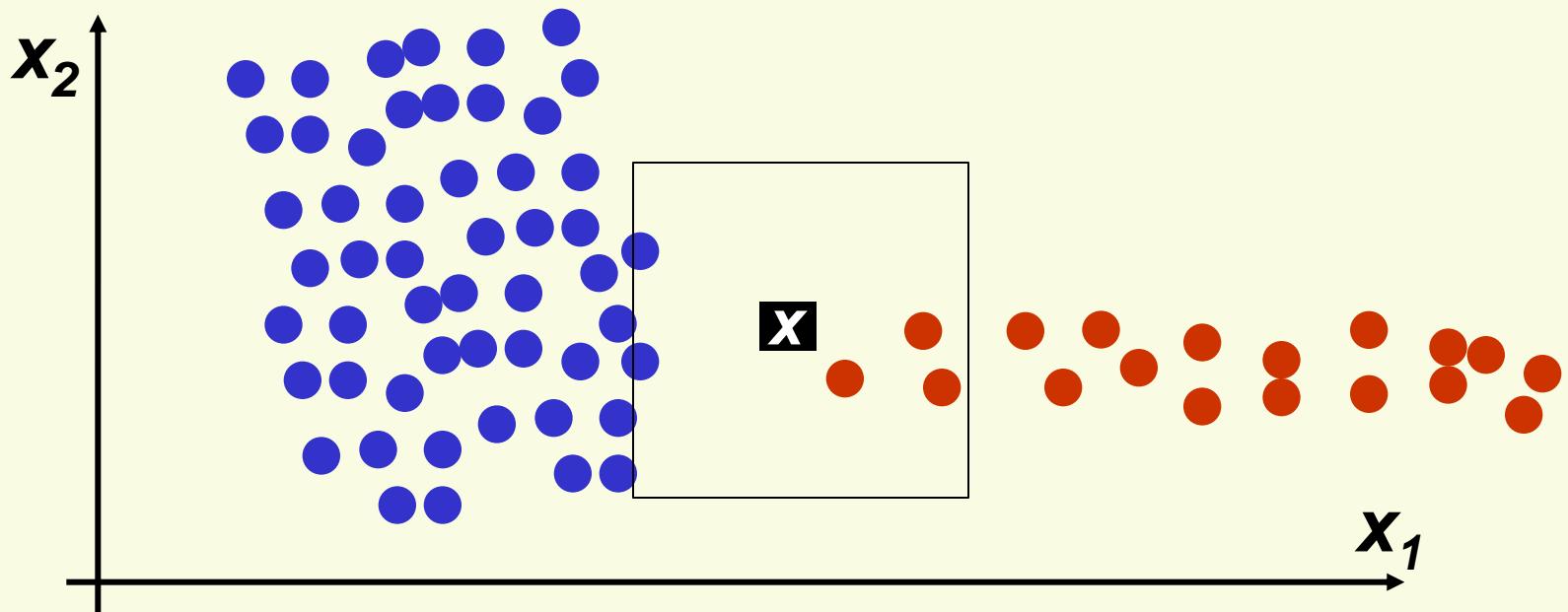
kNN: How to Choose k ?

- In theory, when the infinite number of samples is available, the larger the k , the better is classification (error rate gets closer to the optimal Bayes error rate)
- But the caveat is that all k neighbors have to be close to x
 - Possible when infinite # samples available
 - Impossible in practice since # samples is finite

kNN: How to Choose k?

- In practice
 1. k should be large so that error rate is minimized
 - k too small will lead to noisy decision boundaries
 2. k should be small enough so that only nearby samples are included
 - k too large will lead to over-smoothed boundaries
- Balancing 1 and 2 is not trivial
 - This is a recurrent issue, need to smooth data, but not too much

kNN: How to Choose k?



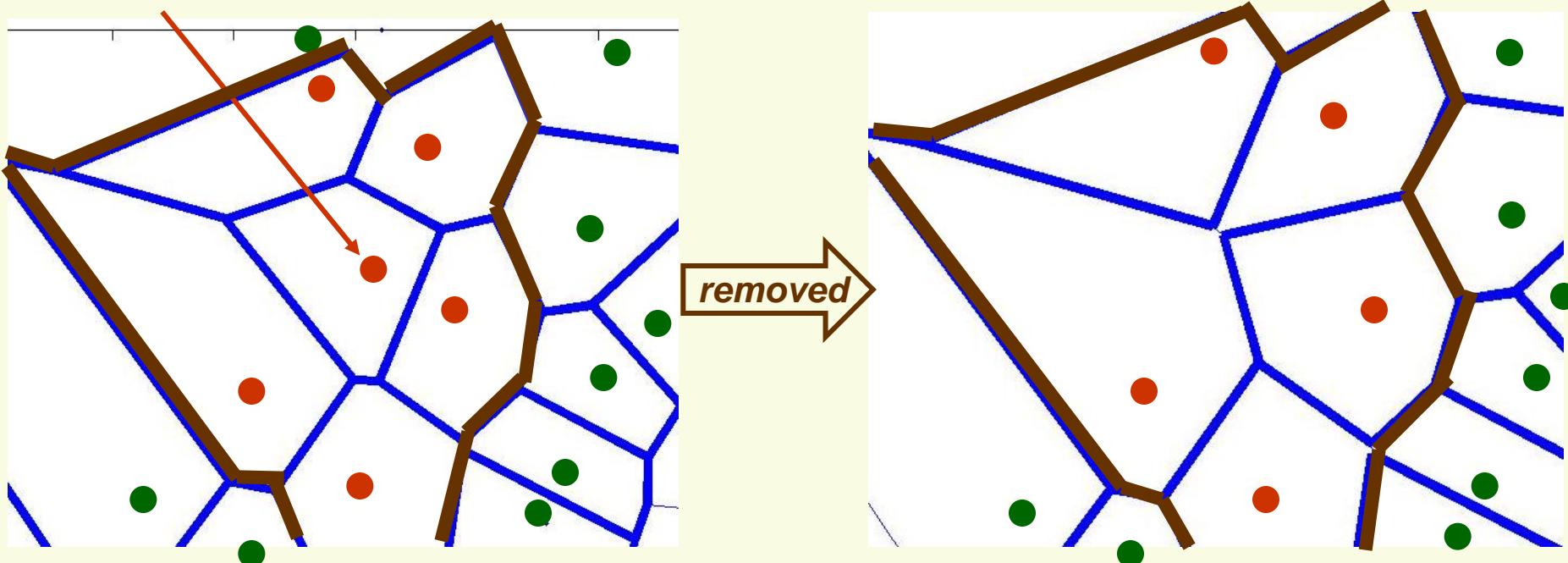
- For $k = 1, \dots, 5$ point x gets classified correctly
 - red class
- For larger k classification of x is wrong
 - blue class

kNN: Computational Complexity

- Basic ***kNN*** algorithm stores all examples. Suppose we have n examples each of dimension d
 - $O(d)$ to compute distance to one example
 - $O(nd)$ to find one nearest neighbor
 - $O(knd)$ to find k closest examples examples
 - Thus complexity is $O(knd)$
- This is prohibitively expensive for large number of samples
- But we need large number of samples for ***kNN*** to work well!

Reducing Complexity: Editing 1NN

- If all voronoi neighbors have the same class, a sample is useless, we can remove it:



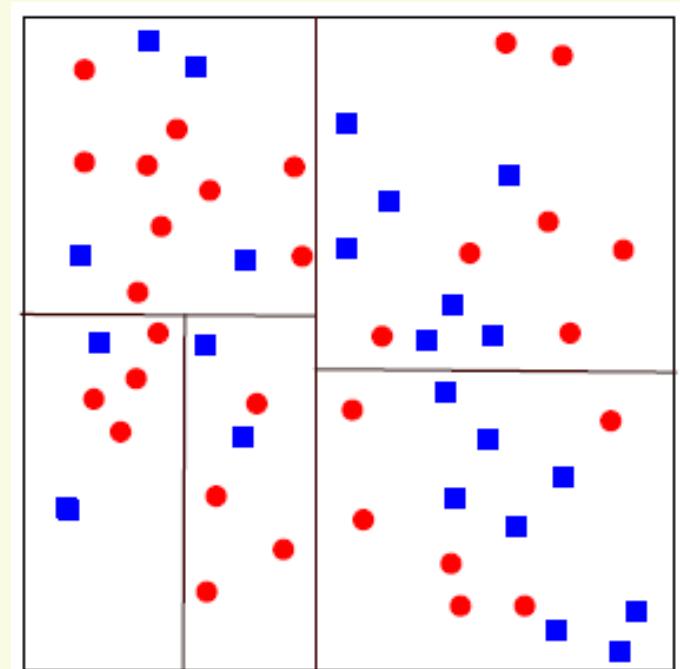
- Number of samples decreases
- We are guaranteed that the decision boundaries stay the same

Reducing the complexity of KNN

- Idea: Partition space recursively and search for NN only close to the test point
- Preprocessing: Done prior to classification process.

Axis-parallel tree construction:

1. Split space in direction of largest ‘spread’ into two equi-numbered cells
2. Repeat procedure recursively for each subcell, until some stopping criterion is achieved



Reducing the complexity of KNN

- Classification:
 1. Propagate a test point down the tree. Classification is based on NN from the final leaf reached.
 2. If NN (within leaf) is further than nearest boundary - retrack
- Notes:
 - Clearly $\log n$ layers (and distance computations) suffice.
 - Computation time to build tree: $O(dn \log n)$ (offline)
 - Many variations and improvements exist (e.g. diagonal splits)
 - Stopping criterion: often ad-hoc (e.g. number of points in leaf region is k , region size, etc.)

kNN: Selection of Distance

- So far we assumed we use Euclidian Distance to find the nearest neighbor:

$$D(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_k (\mathbf{a}_k - \mathbf{b}_k)^2}$$

- However some features (dimensions) may be much more discriminative than other features (dimensions)
- Euclidian distance treats each feature as equally important

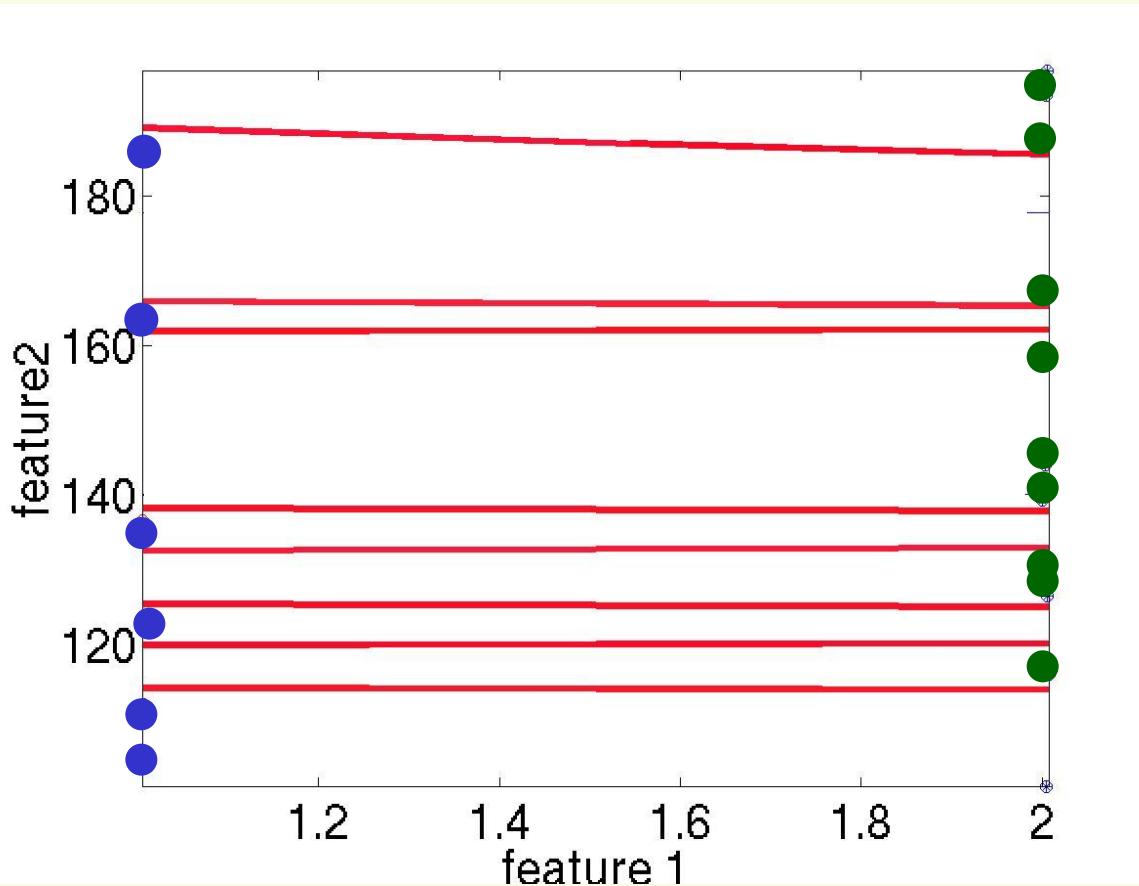
kNN: Selection of Distance

- Extreme Example
 - feature 1 gives the correct class: 1 or 2
 - feature 2 gives irrelevant number from 100 to 200
- Suppose we have to find the class of $x=[1 \ 100]$ and we have 2 samples $[1 \ 150]$ and $[2 \ 110]$

$$D\left(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 1 \\ 150 \end{bmatrix}\right) = \sqrt{(1-1)^2 + (100-150)^2} = 50 \quad D\left(\begin{bmatrix} 1 \\ 100 \end{bmatrix}, \begin{bmatrix} 2 \\ 110 \end{bmatrix}\right) = \sqrt{(1-2)^2 + (100-110)^2} = 10.5$$

- $x = [1 \ 100]$ is misclassified!
- The denser the samples, the less of the problem
 - But we rarely have samples dense enough

kNN: Extreme Example of Distance Selection

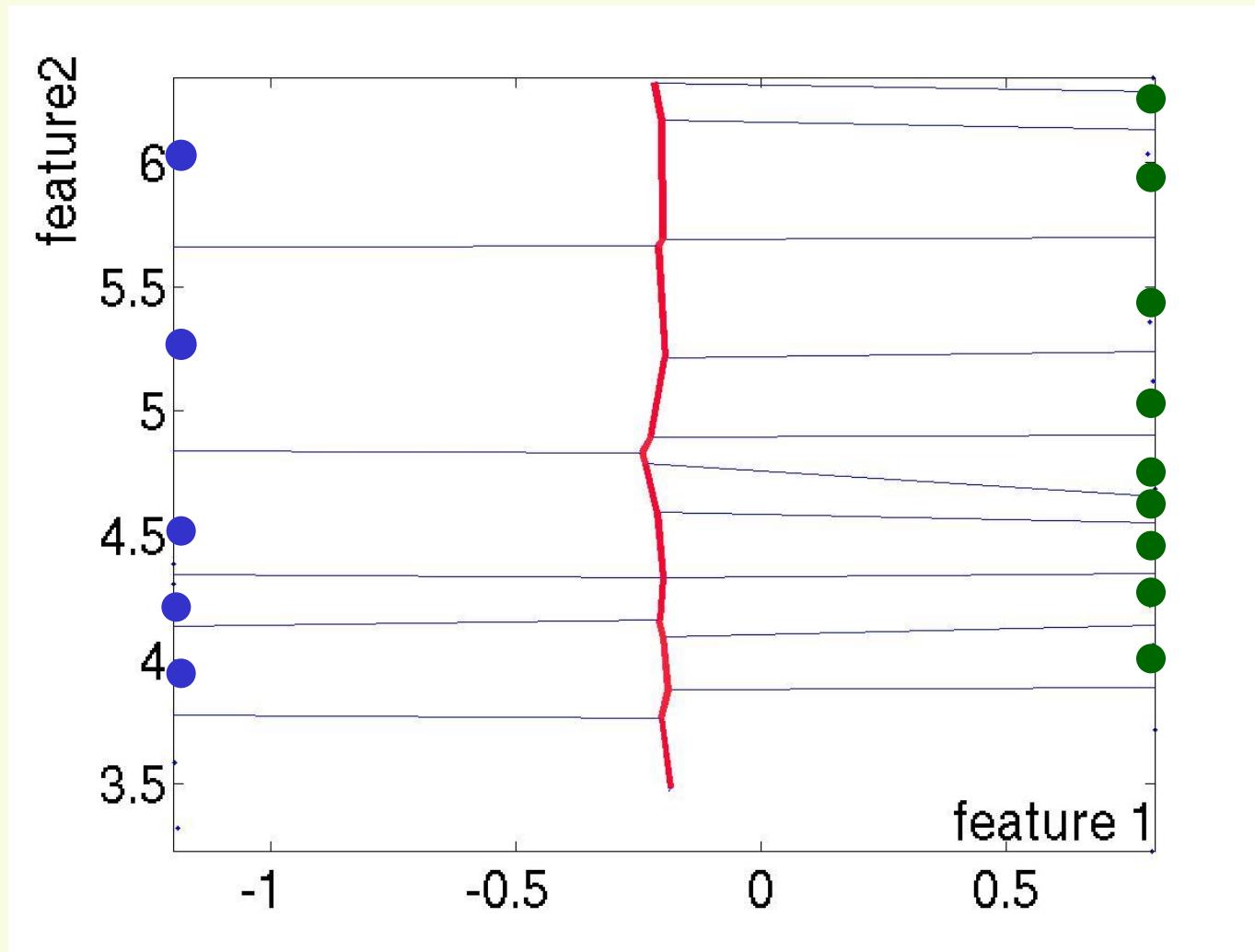


- decision boundaries for blue and green classes are in red
- These boundaries are really bad because
 - feature 1 is discriminative, but its scale is small
 - feature 2 gives no class information (noise) but its scale is large

kNN: Selection of Distance

- Notice the 2 features are on different scales:
 - feature 1 takes values between 1 or 2
 - feature 2 takes values between 100 to 200
- We could normalize each feature to be between of mean 0 and variance 1
- If X is a random variable of mean μ and variance σ^2 , then $(X - \mu)/\sigma$ has mean 0 and variance 1
- Thus for each feature vector x_i , compute its sample mean and variance, and let the new feature be $[x_i - \text{mean}(x_i)]/\sqrt{\text{var}(x_i)}$
- Let's do it in the previous example

kNN: Normalized Features



- The decision boundary (in red) is very good now!

kNN: Selection of Distance

- However in high dimensions if there are a lot of irrelevant features, normalization will not help

$$D(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_k (\mathbf{a}_k - \mathbf{b}_k)^2} = \sqrt{\sum_i (\mathbf{a}_i - \mathbf{b}_i)^2 + \sum_j (\mathbf{a}_j - \mathbf{b}_j)^2}$$

discriminative feature *noisy features*

- If the number of discriminative features is smaller than the number of noisy features, Euclidean distance is dominated by noise

kNN: Feature Weighting

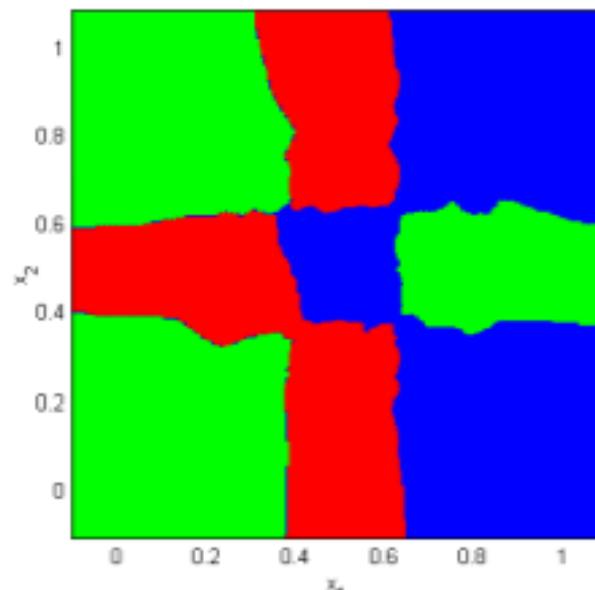
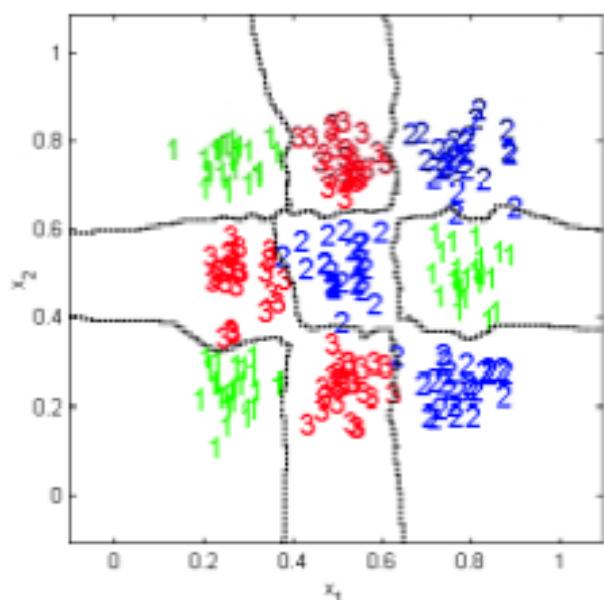
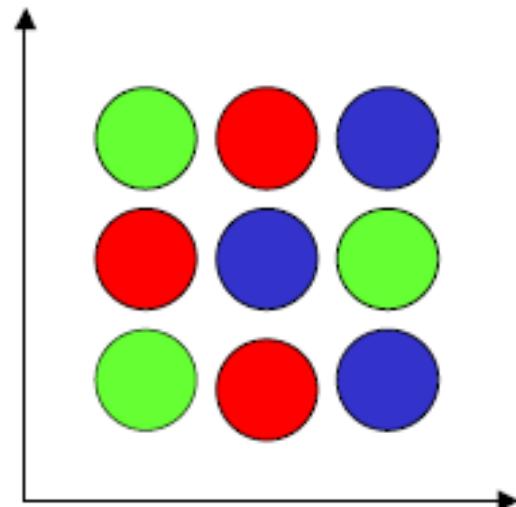
- Scale each feature by its importance for classification

$$D(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_k w_k (a_k - b_k)^2}$$

- Can learn the weights w_k from the validation data
 - Increase/decrease weights until classification improves

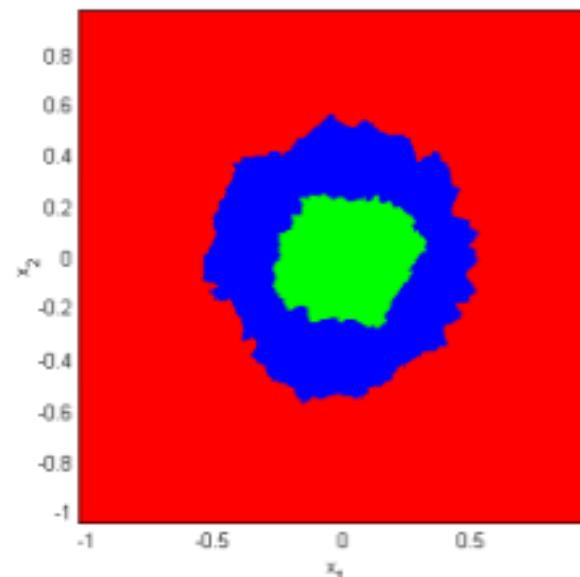
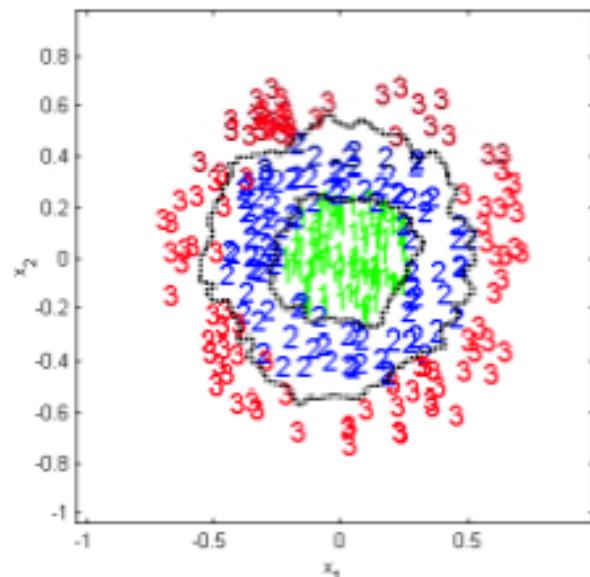
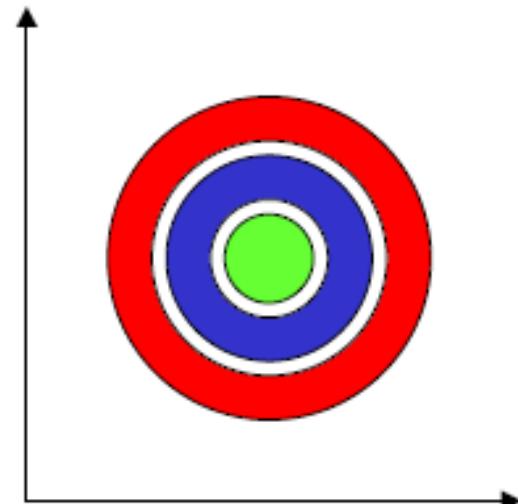
k-NNR in action: example 1

- We have generated data for a 2-dimensional 3-class problem, where the class-conditional densities are multi-modal, and non-linearly separable, as illustrated in the figure
- We used the k-NNR with
 - $k = 5$
 - Metric = Euclidean distance
- The resulting decision boundaries and decision regions are shown below



k-NNR in action: example 2

- We have generated data for a 2-dimensional 3-class problem, where the class-conditional densities are unimodal, and are distributed in rings around a common mean. These classes are also non-linearly separable, as illustrated in the figure
- We used the k-NNR with
 - $k = five$
 - Metric = Euclidean distance
- The resulting decision boundaries and decision regions are shown below



kNN Summary

- Advantages
 - Can be applied to the data from any distribution
 - Very simple and intuitive
 - Good classification if the number of samples is large enough
- Disadvantages
 - Choosing best k may be difficult
 - Computationally heavy, but improvements possible
 - Need large number of samples for accuracy
 - Can never fix this without assuming parametric distribution

Curse of Dimensionality, Dimensionality Reduction with PCA

Curse of Dimensionality: Overfitting

- If the number of features d is large, the number of samples n , may be too small for accurate parameter estimation.
- For example, covariance matrix has d^2 parameters:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \cdots & \sigma_{1d} \\ \vdots & \ddots & \vdots \\ \sigma_{d1} & \cdots & \sigma_d^2 \end{bmatrix}$$

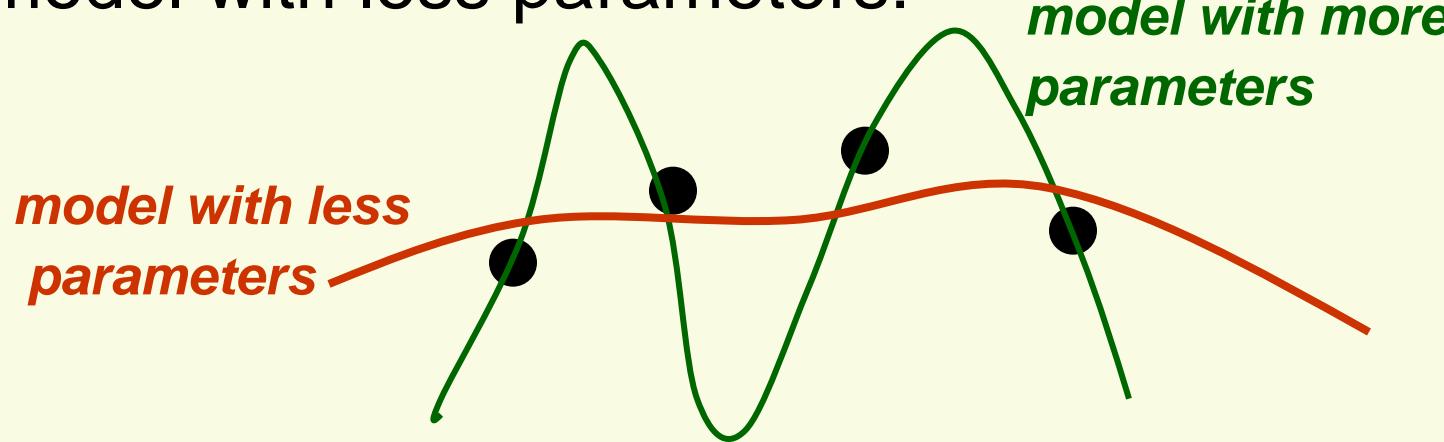
- For accurate estimation, n should be much bigger than d^2 , otherwise model is too complicated for the data, **overfitting**:

Curse of Dimensionality: Overfitting

- Paradox: If $n < d^2$ we are better off assuming that features are uncorrelated, even if we know this assumption is wrong
- In this case, the covariance matrix has only d parameters:

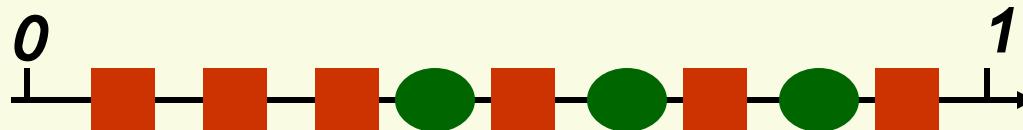
$$\Sigma = \begin{bmatrix} \sigma_1^2 & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \sigma_d^2 \end{bmatrix}$$

- We are likely to avoid overfitting because we fit a model with less parameters:



Curse of Dimensionality: Number of Samples

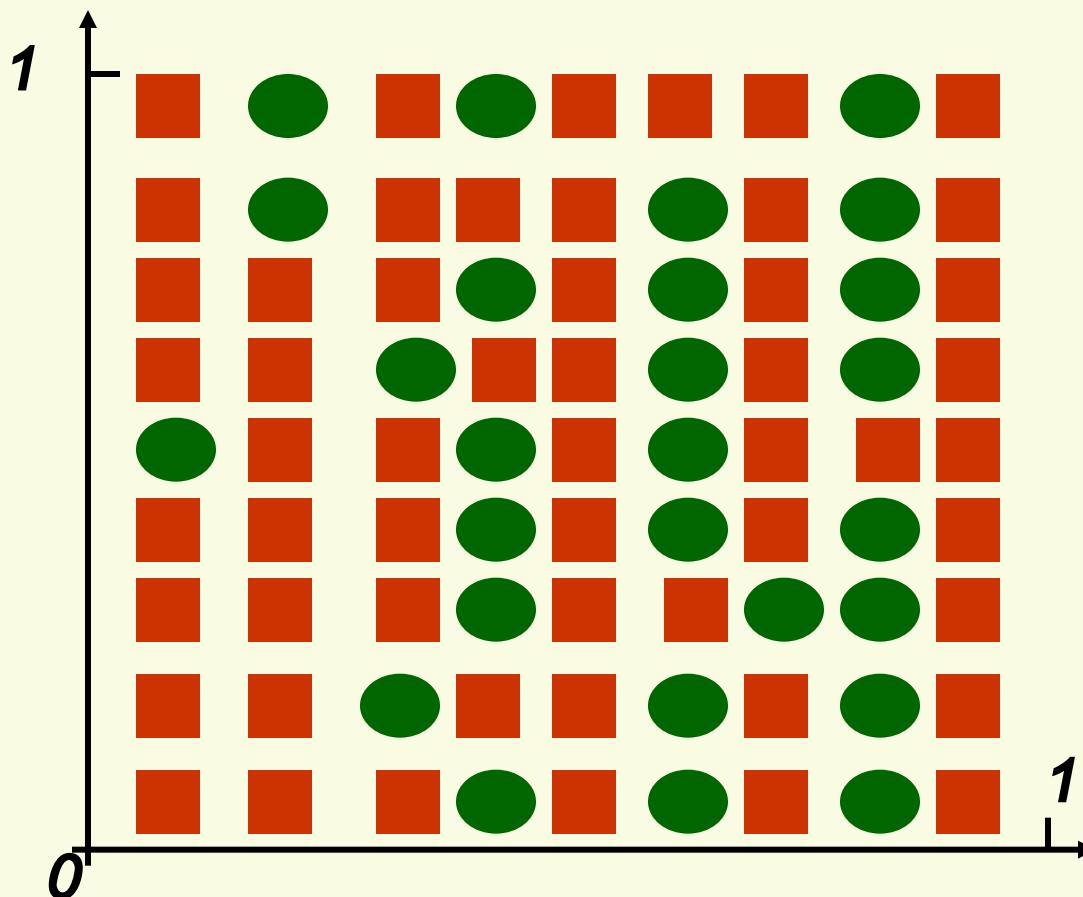
- Suppose we want to use the nearest neighbor approach with $k = 1$ (**1NN**)
- Suppose we start with only one feature



- This feature is not discriminative, i.e. it does not separate the classes well
- We decide to use 2 features. For the 1NN method to work well, need a lot of samples, i.e. samples have to be dense
- To maintain the same density as in 1D (9 samples per unit length), how many samples do we need?

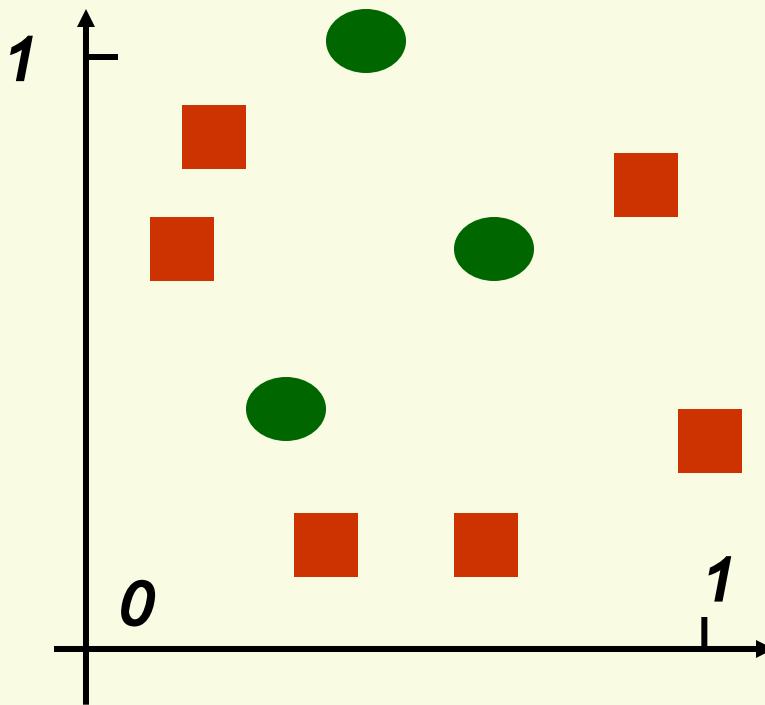
Curse of Dimensionality: Number of Samples

- We need 9^2 samples to maintain the same density as in $1D$



Curse of Dimensionality: Number of Samples

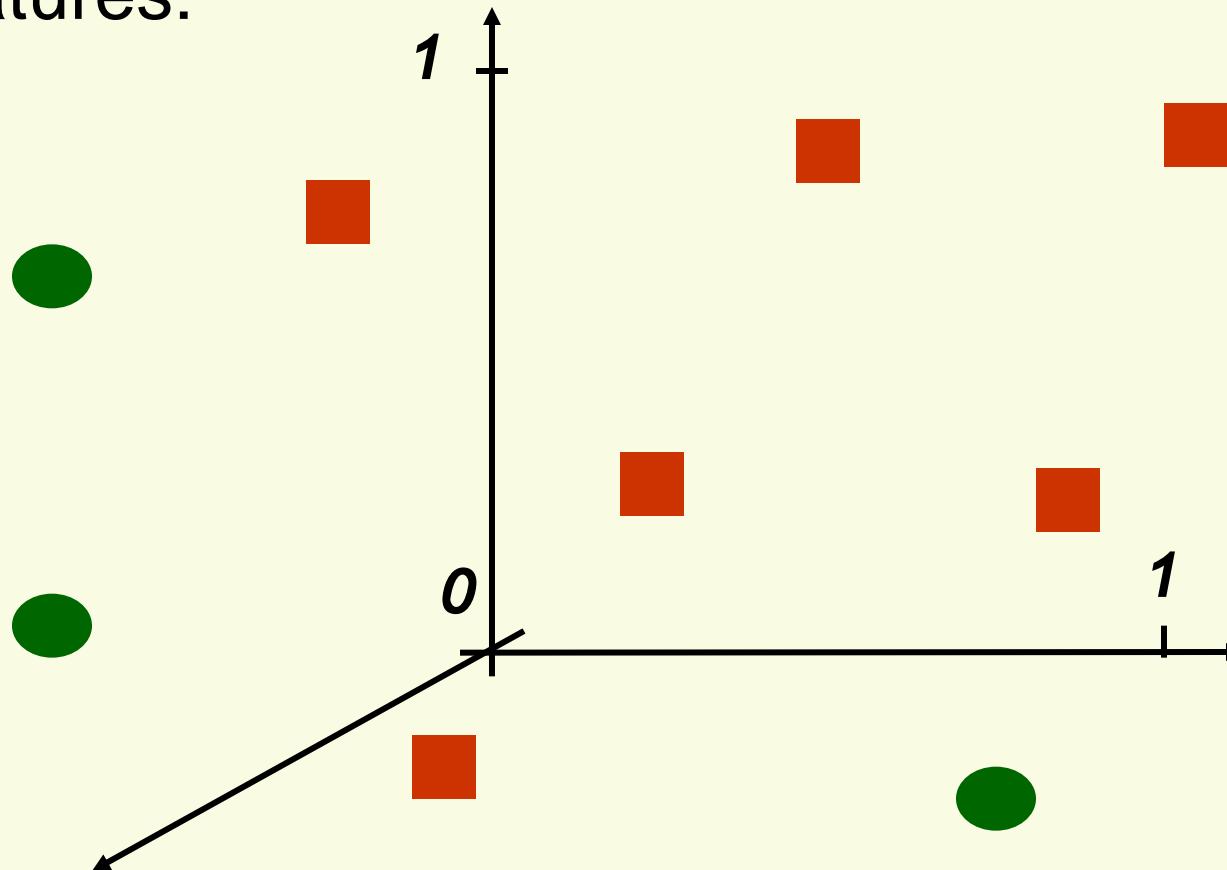
- Of course, when we go from 1 feature to 2, no one gives us more samples, we still have 9



- This is way too sparse for **1NN** to work well

Curse of Dimensionality: Number of Samples

- Things go from bad to worse if we decide to use 3 features:



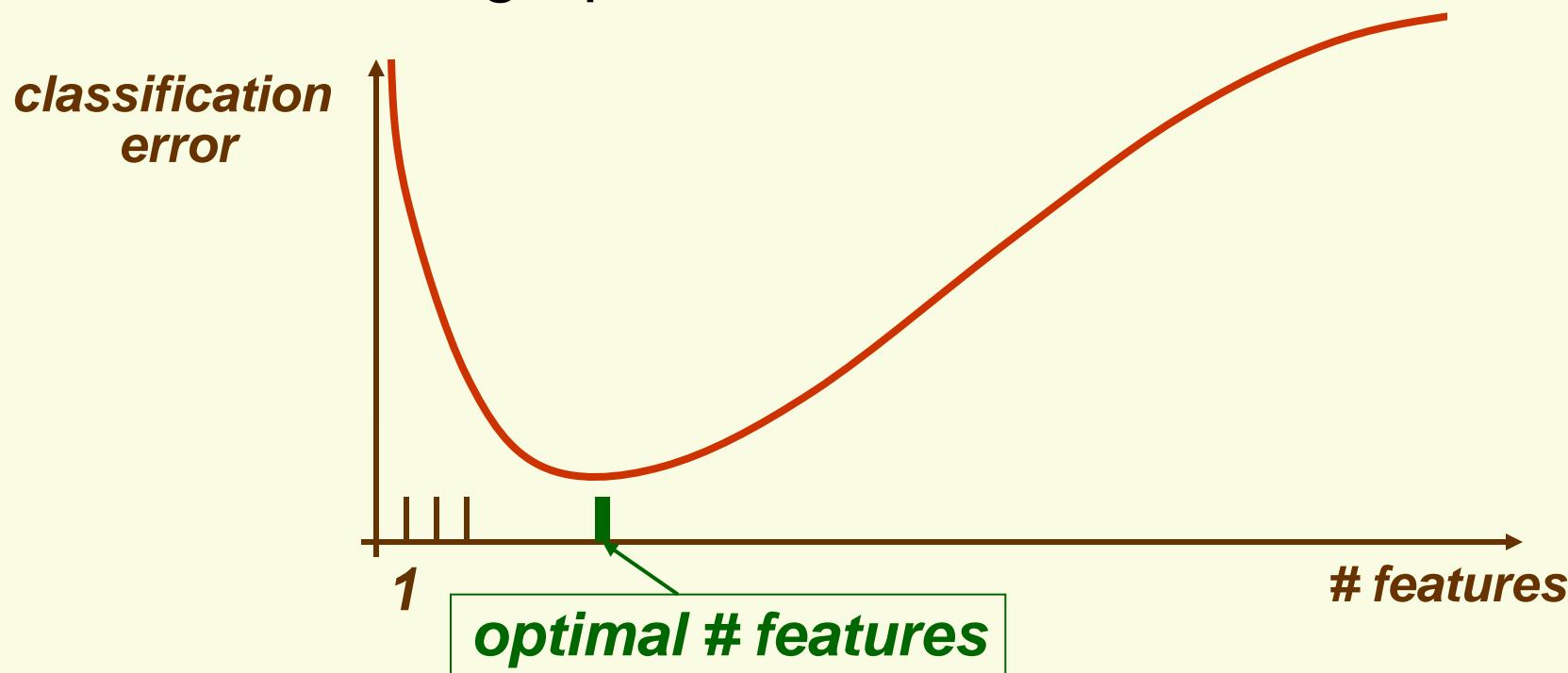
- If **9** was dense enough in 1D, in 3D we need $9^3 = 729$ samples!

Curse of Dimensionality: Number of Samples

- In general, if n samples is dense enough in **$1D$**
- Then in d dimensions we need n^d samples!
- And n^d grows really really fast as a function of d
- Common pitfall:
 - If we can't solve a problem with a few features, adding more features seems like a good idea
 - However the number of samples usually stays the same
 - The method with more features is likely to perform worse instead of expected better

Curse of Dimensionality: Number of Samples

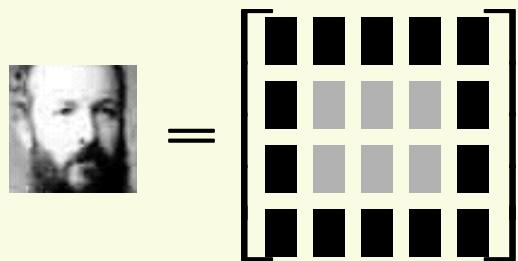
- For a fixed number of samples, as we add features, the graph of classification error:



- Thus for each fixed sample size n , there is the optimal number of features to use

The Curse of Dimensionality

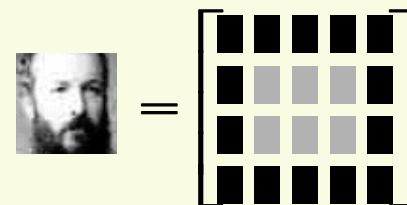
- We should try to avoid creating lot of features
- Often no choice, problem starts with many features
- Example: Face Detection
 - One sample point is k by m array of pixels


$$\text{Portrait} = \begin{bmatrix} \text{Black} & \text{Black} & \text{Black} & \text{Black} & \text{Black} \\ \text{Black} & \text{Grey} & \text{Grey} & \text{Grey} & \text{Black} \\ \text{Black} & \text{Grey} & \text{Grey} & \text{Grey} & \text{Black} \\ \text{Black} & \text{Black} & \text{Black} & \text{Black} & \text{Black} \end{bmatrix}$$

- Feature extraction is not trivial, usually every pixel is taken as a feature
- Typical dimension is 20 by $20 = 400$
- Suppose 10 samples are dense enough for 1 dimension. Need only 10^{400} samples

The Curse of Dimensionality

- Face Detection, dimension of one sample point is ***km***

A small portrait of a man with a beard is positioned to the left of an equals sign (=). To the right of the equals sign is a 4x4 matrix of colored squares. The first column contains four black squares. The second column contains three black squares and one light gray square. The third column contains two black squares and two light gray squares. The fourth column contains three black squares and one light gray square.

- The fact that we set up the problem with ***km*** dimensions (features) does not mean it is really a ***km***-dimensional problem
- Space of all ***k*** by ***m*** images has ***km*** dimensions
- Space of all ***k*** by ***m*** faces must be much smaller, since faces form a tiny fraction of all possible images
- Most likely we are not setting the problem up with the right features
- If we used better features, we are likely need much less than ***km***-dimensions

Dimensionality Reduction

- High dimensionality is challenging and redundant
- It is natural to try to reduce dimensionality
- Reduce dimensionality by feature combination:
combine old features \mathbf{x} to create new features \mathbf{y}

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_d \end{bmatrix} \rightarrow f\left(\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_d \end{bmatrix}\right) = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_k \end{bmatrix} = \mathbf{y} \quad \text{with } k < d$$

- For example,
- $$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{x}_1 + \mathbf{x}_2 \\ \mathbf{x}_3 + \mathbf{x}_4 \end{bmatrix} = \mathbf{y}$$
- Ideally, the new vector \mathbf{y} should retain from \mathbf{x} all information important for classification

Dimensionality Reduction

- The best $f(\mathbf{x})$ is most likely a non-linear function
- Linear functions are easier to find though
- For now, assume that $f(\mathbf{x})$ is a linear mapping
- Thus it can be represented by a matrix \mathbf{W} :

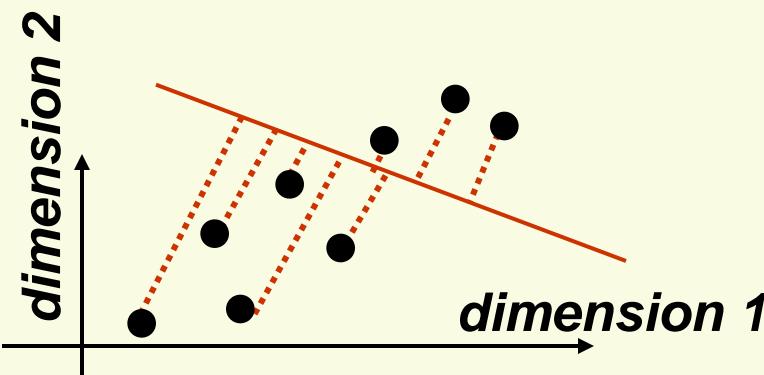
$$\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_d \end{bmatrix} \Rightarrow \mathbf{W} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_d \end{bmatrix} = \begin{bmatrix} \mathbf{w}_{11} & \cdots & \mathbf{w}_{1d} \\ \vdots & & \vdots \\ \mathbf{w}_{k1} & \cdots & \mathbf{w}_{kd} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_d \end{bmatrix} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_k \end{bmatrix} \quad \text{with } k < d$$

Feature Combination

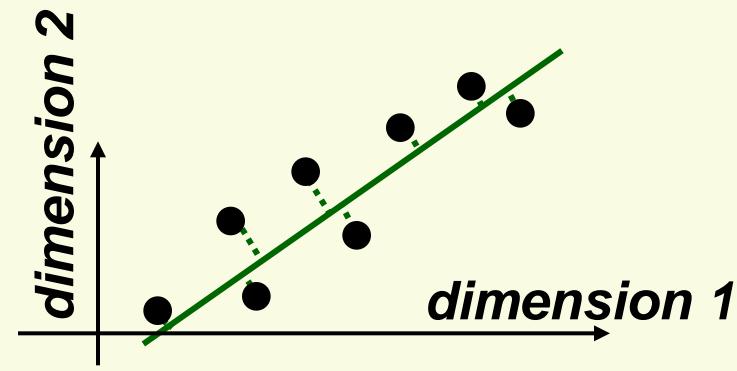
- We will look at 2 methods for feature combination
 - Principle Component Analysis (PCA)
 - Fischer Linear Discriminant (next lecture)

Principle Component Analysis (PCA)

- Main idea: seek most accurate data representation in a lower dimensional space
- Example in 2-D
 - Project data to 1-D subspace (a line) which minimize the projection error



*large projection errors,
bad line to project to*

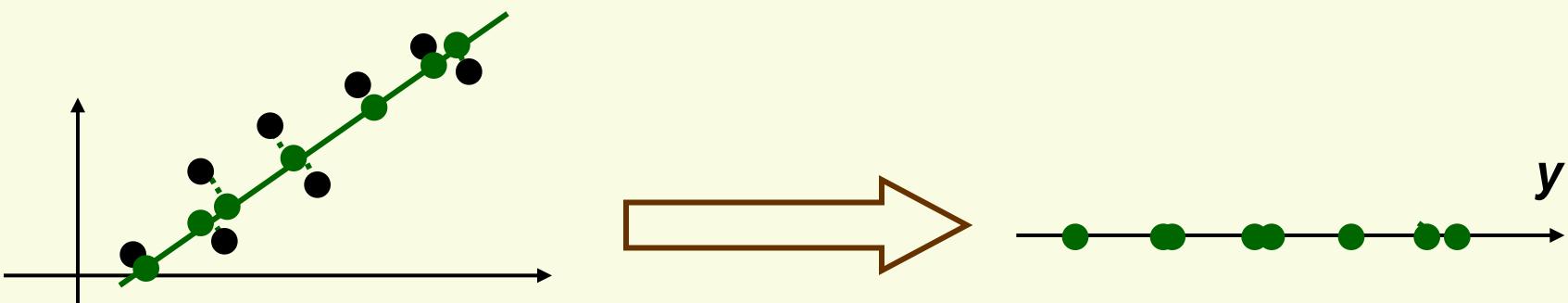


*small projection errors,
good line to project to*

- Notice that the good line to use for projection lies in the direction of largest variance

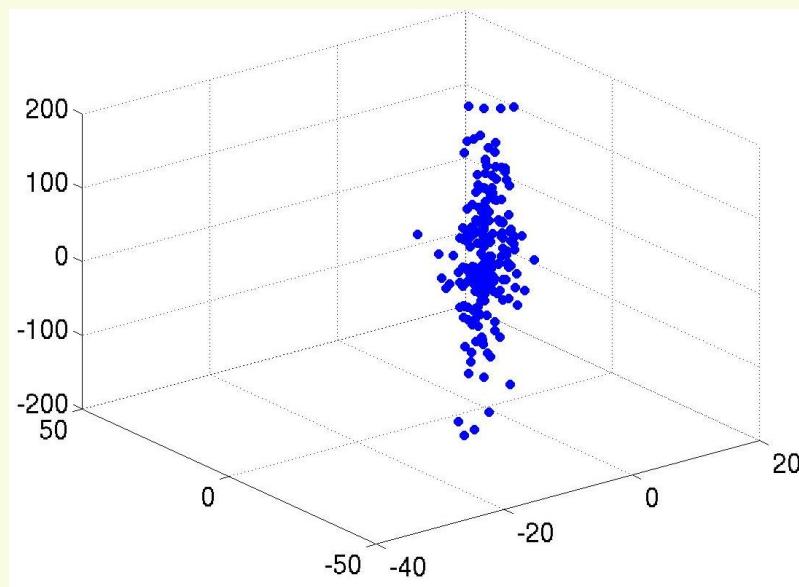
PCA

- After the data is projected on the best line, need to transform the coordinate system to get 1D representation for vector y

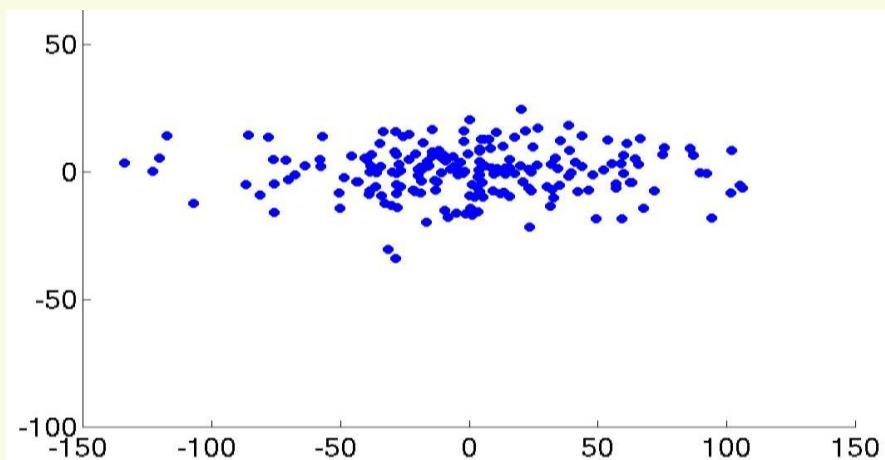


- Note that new data y has the same variance as old data x in the direction of the green line
- PCA preserves largest variances in the data. We will prove this statement, for now it is just an intuition of what PCA will do

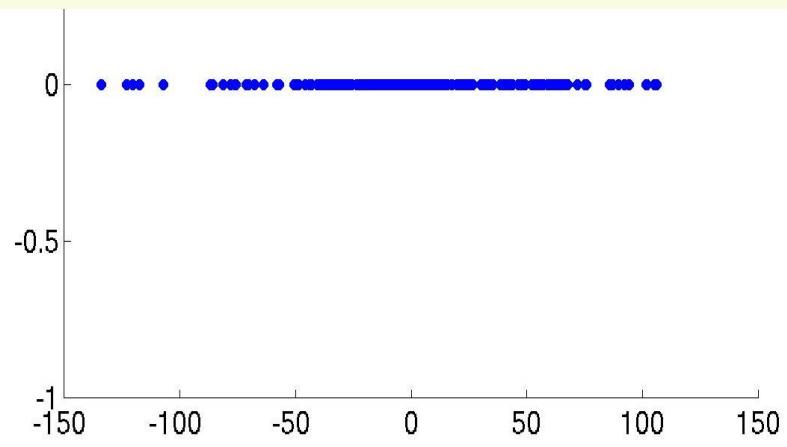
PCA: Approximation of Elliptical Cloud in 3D



best 2D approximation



best 1D approximation



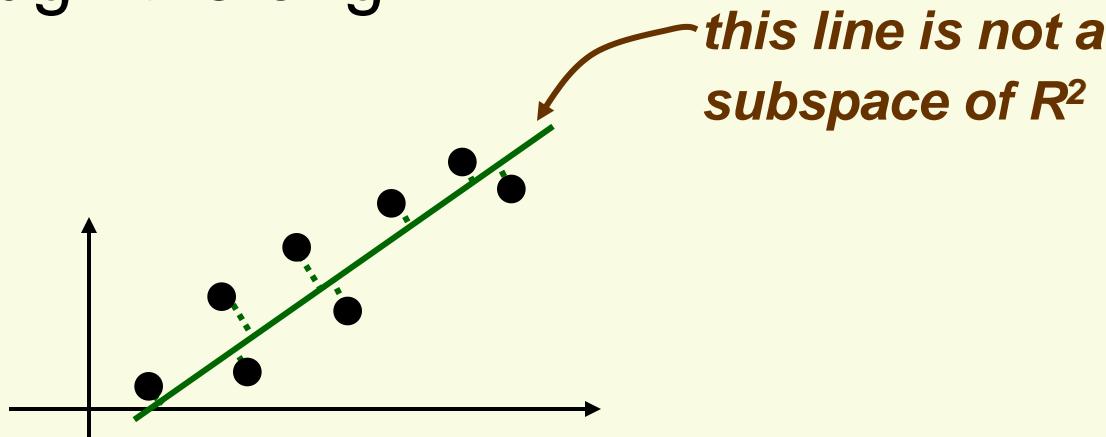
PCA: Linear Algebra for Derivation

- Let V be a d dimensional linear space, and W be a k dimensional linear subspace of V
- We can always find a set of d dimensional vectors $\{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_k\}$ which forms an orthonormal basis for W
 - $\langle \mathbf{e}_i, \mathbf{e}_j \rangle = 0$ if i is not equal to j and $\langle \mathbf{e}_i, \mathbf{e}_i \rangle = 1$
- Thus any vector in W can be written as

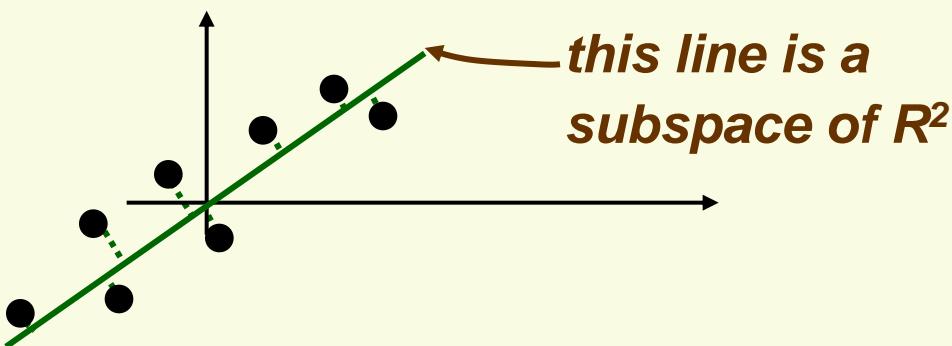
$$\alpha_1 \mathbf{e}_1 + \alpha_2 \mathbf{e}_2 + \dots + \alpha_k \mathbf{e}_k = \sum_{i=1}^k \alpha_i \mathbf{e}_i \quad \text{for scalars } \alpha_1, \dots, \alpha_k$$

PCA: Linear Algebra for Derivation

- Recall that subspace W contains the zero vector, i.e. it goes through the origin



- For derivation, it will be convenient to project to subspace W : thus we need to shift everything

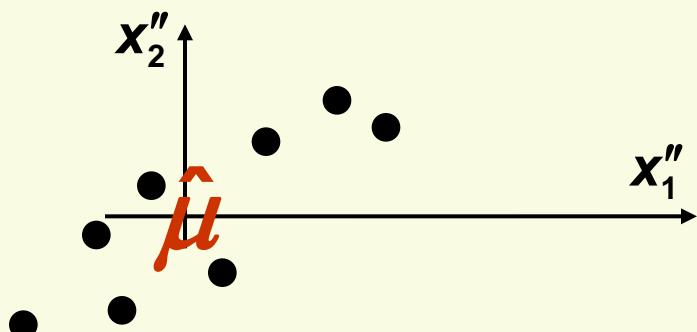
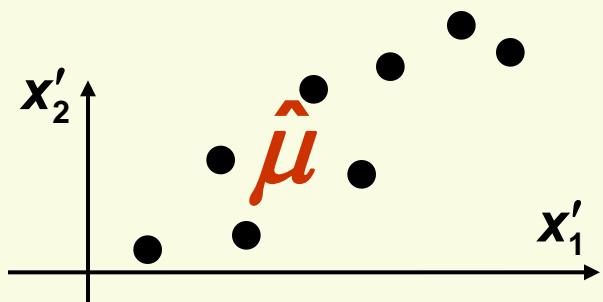


PCA Derivation: Shift by the Mean Vector

- Before PCA, subtract sample mean from the data

$$\mathbf{x} - \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i = \mathbf{x} - \hat{\boldsymbol{\mu}}$$

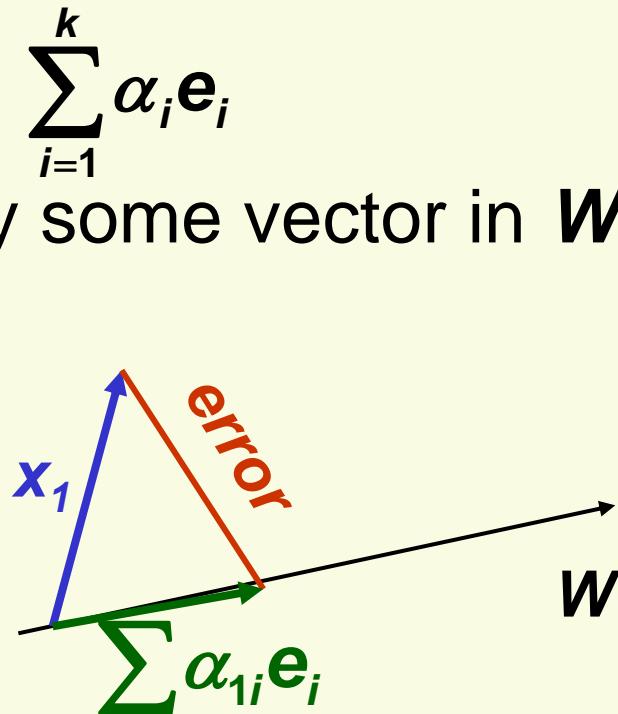
- The new data has zero mean.
- All we did is change the coordinate system



PCA: Derivation

- We want to find the most accurate representation of data $D=\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ in some subspace W which has dimension $k < d$
- Let $\{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_k\}$ be the orthonormal basis for W . Any vector in W can be written as $\sum_{i=1}^k \alpha_i \mathbf{e}_i$
- Thus \mathbf{x}_1 will be represented by some vector in W $\sum_{i=1}^k \alpha_{1i} \mathbf{e}_i$
- Error of this representation:

$$\text{error} = \left\| \mathbf{x}_1 - \sum_{i=1}^k \alpha_{1i} \mathbf{e}_i \right\|^2$$



PCA: Derivation

- To find the total error, we need to sum over all x_j 's
- Any x_j can be written as $\sum_{i=1}^k \alpha_{ji} e_i$
- Thus the total error for representation of all data D is:

sum over all data points

$$J(\underbrace{e_1, \dots, e_k, \alpha_{11}, \dots, \alpha_{nk}}_{\text{unknowns}}) = \sum_{j=1}^n \left\| x_j - \sum_{i=1}^k \alpha_{ji} e_i \right\|^2$$

error at one point

PCA: Derivation

- To minimize J , need to take partial derivatives and also enforce constraint that $\{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_k\}$ are orthogonal

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k, \alpha_{11}, \dots, \alpha_{nk}) = \sum_{j=1}^n \left\| \mathbf{x}_j - \sum_{i=1}^k \alpha_{ji} \mathbf{e}_i \right\|^2$$

- Let us simplify J first:

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k, \alpha_{11}, \dots, \alpha_{nk}) = \sum_{j=1}^n \left\| \mathbf{x}_j \right\|^2 - 2 \sum_{j=1}^n \sum_{i=1}^k \alpha_{ji} \mathbf{x}_j^t \mathbf{e}_i + \sum_{j=1}^n \sum_{i=1}^k \alpha_{ji}^2$$

PCA: Derivation

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k, \alpha_{11}, \dots, \alpha_{nk}) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - 2 \sum_{j=1}^n \sum_{i=1}^k \alpha_{ji} \mathbf{x}_j^t \mathbf{e}_i + \sum_{j=1}^n \sum_{i=1}^k \alpha_{ji}^2$$

- First take partial derivatives with respect to α_{ml}

$$\frac{\partial}{\partial \alpha_{ml}} J(\mathbf{e}_1, \dots, \mathbf{e}_k, \alpha_{11}, \dots, \alpha_{nk}) = -2 \mathbf{x}_m^t \mathbf{e}_l + 2\alpha_{ml}$$

- Thus the optimal value for α_{ml} is

$$-2 \mathbf{x}_m^t \mathbf{e}_l + 2\alpha_{ml} = 0 \Rightarrow \alpha_{ml} = \mathbf{x}_m^t \mathbf{e}_l$$

PCA: Derivation

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k, \alpha_{11}, \dots, \alpha_{nk}) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - 2 \sum_{j=1}^n \sum_{i=1}^k \alpha_{ji} \mathbf{x}_j^t \mathbf{e}_i + \sum_{j=1}^n \sum_{i=1}^k \alpha_{ji}^2$$

- Plug the optimal value for $\alpha_{ml} = \mathbf{x}_m^t \mathbf{e}_l$ back into J

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - 2 \sum_{j=1}^n \sum_{i=1}^k (\mathbf{x}_j^t \mathbf{e}_i) \mathbf{x}_j^t \mathbf{e}_i + \sum_{j=1}^n \sum_{i=1}^k (\mathbf{x}_j^t \mathbf{e}_i)^2$$

- Can simplify J

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{j=1}^n \sum_{i=1}^k (\mathbf{x}_j^t \mathbf{e}_i)^2$$

PCA: Derivation

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{j=1}^n \sum_{i=1}^k (\mathbf{x}_j^t \mathbf{e}_i)^2$$

- Rewrite J using $(\mathbf{a}^t \mathbf{b})^2 = (\mathbf{a}^t \mathbf{b})(\mathbf{a}^t \mathbf{b}) = (\mathbf{b}^t \mathbf{a})(\mathbf{a}^t \mathbf{b}) = \mathbf{b}^t (\mathbf{a} \mathbf{a}^t) \mathbf{b}$

$$\begin{aligned} J(\mathbf{e}_1, \dots, \mathbf{e}_k) &= \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{i=1}^k \mathbf{e}_i^t \left(\sum_{j=1}^n (\mathbf{x}_j \mathbf{x}_j^t) \right) \mathbf{e}_i \\ &= \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{i=1}^k \mathbf{e}_i^t \mathbf{S} \mathbf{e}_i \end{aligned}$$

- Where $\mathbf{S} = \sum_{j=1}^n \mathbf{x}_j \mathbf{x}_j^t$
- \mathbf{S} is called the scatter matrix, it is just $n-1$ times the sample covariance matrix we have seen before

$$\hat{\Sigma} = \frac{1}{n-1} \sum_{j=1}^n (\mathbf{x}_j - \hat{\mu})(\mathbf{x}_j - \hat{\mu})^t$$

PCA: Derivation

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{i=1}^k \mathbf{e}_i^t \mathbf{S} \mathbf{e}_i$$

constant

- Minimizing J is equivalent to maximizing $\sum_{i=1}^k \mathbf{e}_i^t \mathbf{S} \mathbf{e}_i$,
- We should also enforce constraints $\mathbf{e}_i^t \mathbf{e}_i = 1$ for all i
- Use the method of Lagrange multipliers, incorporate the constraints with undetermined $\lambda_1, \dots, \lambda_k$
- Need to maximize new function u

$$u(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{i=1}^k \mathbf{e}_i^t \mathbf{S} \mathbf{e}_i - \sum_{j=1}^k \lambda_j (\mathbf{e}_j^t \mathbf{e}_j - 1)$$

PCA: Derivation

$$u(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{i=1}^k \mathbf{e}_i^t \mathbf{S} \mathbf{e}_i - \sum_{j=1}^k \lambda_j (\mathbf{e}_j^t \mathbf{e}_j - 1)$$

- Compute the partial derivatives with respect to \mathbf{e}_m

$$\frac{\partial}{\partial \mathbf{e}_m} u(\mathbf{e}_1, \dots, \mathbf{e}_k) = 2\mathbf{S}\mathbf{e}_m - 2\lambda_m \mathbf{e}_m = 0$$

Note: \mathbf{e}_m is a vector, what we are really doing here is taking partial derivatives with respect to each element of \mathbf{e}_m and then arranging them up in a linear equation

- Thus λ_m and \mathbf{e}_m are eigenvalues and eigenvectors of scatter matrix \mathbf{S}

$$\mathbf{S}\mathbf{e}_m = \lambda_m \mathbf{e}_m$$

PCA: Derivation

$$J(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{i=1}^k \mathbf{e}_i^t \mathbf{S} \mathbf{e}_i$$

- Let's plug \mathbf{e}_m back into J and use $\mathbf{S}\mathbf{e}_m = \lambda_m \mathbf{e}_m$

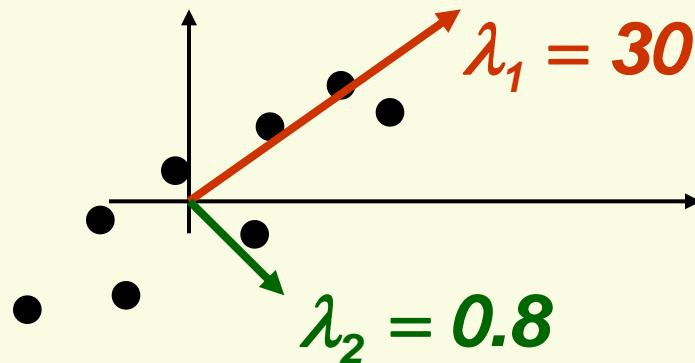
$$J(\mathbf{e}_1, \dots, \mathbf{e}_k) = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{i=1}^k \lambda_i \|\mathbf{e}_i\|^2 = \sum_{j=1}^n \|\mathbf{x}_j\|^2 - \sum_{i=1}^k \lambda_i$$

constant

- Thus to minimize J take for the basis of \mathbf{W} the k eigenvectors of \mathbf{S} corresponding to the k largest eigenvalues

PCA

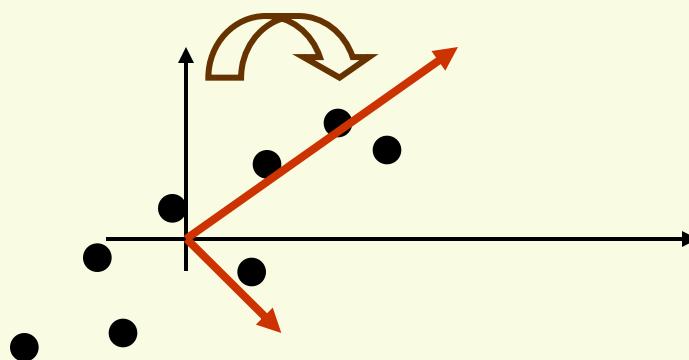
- The larger the eigenvalue of \mathbf{S} , the larger is the variance in the direction of corresponding eigenvector



- This result is exactly what we expected: project \mathbf{x} into subspace of dimension k which has the largest variance
- This is very intuitive: restrict attention to directions where the scatter is the greatest

PCA

- Thus PCA can be thought of as finding new orthogonal basis by rotating the old axis until the directions of maximum variance are found



PCA as Data Approximation

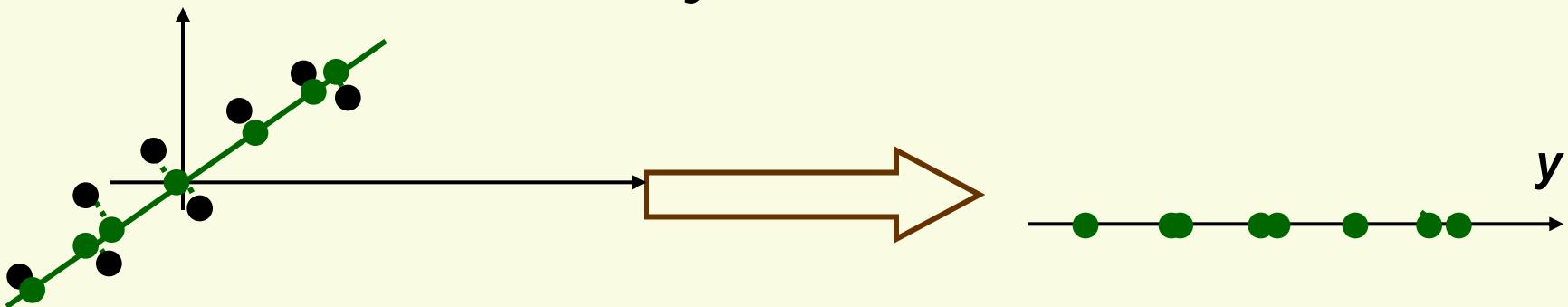
- Let $\{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_d\}$ be all d eigenvectors of the scatter matrix \mathbf{S} , sorted in order of decreasing corresponding eigenvalue
- Without any approximation, for any sample \mathbf{x}_i :

$$\mathbf{x}_i = \sum_{j=1}^d \alpha_j \mathbf{e}_j = \underbrace{\alpha_1 \mathbf{e}_1 + \dots + \alpha_k \mathbf{e}_k}_{\text{approximation of } \mathbf{x}_i} + \overbrace{\alpha_{k+1} \mathbf{e}_{k+1} + \dots + \alpha_d \mathbf{e}_d}^{\text{error of approximation}}$$

- coefficients $\alpha_m = \mathbf{x}_i^t \mathbf{e}_m$ are called *principle components*
 - The larger k , the better is the approximation
 - Components are arranged in order of importance, more important components come first
- Thus PCA takes the first k most important components of \mathbf{x}_i as an approximation to \mathbf{x}_i

PCA: Last Step

- Now we know how to project the data
- Last step is to change the coordinates to get final k -dimensional vector y



- Let matrix $E = [\mathbf{e}_1 \cdots \mathbf{e}_k]$
- Then the coordinate transformation is $y = E^t x$
- Under E^t , the eigenvectors become the standard basis:

$$E^t \mathbf{e}_i = \begin{bmatrix} \mathbf{e}_1 \\ \vdots \\ \mathbf{e}_i \\ \vdots \\ \mathbf{e}_k \end{bmatrix} \mathbf{e}_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Recipe for Dimension Reduction with PCA

Data $\mathbf{D} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. Each \mathbf{x}_i is a d -dimensional vector. Wish to use PCA to reduce dimension to k

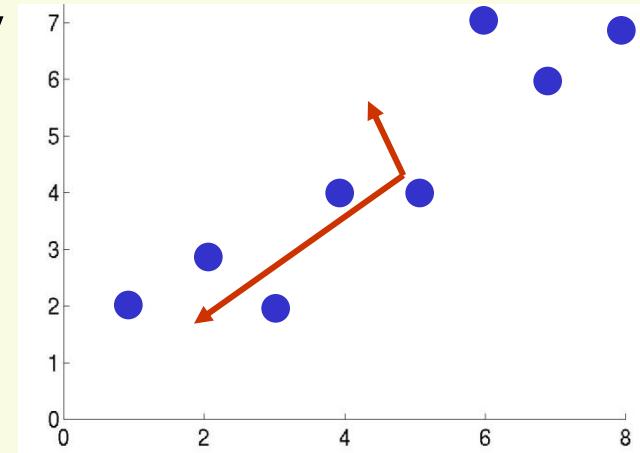
1. Find the sample mean $\hat{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$
2. Subtract sample mean from the data $\mathbf{z}_i = \mathbf{x}_i - \hat{\mu}$
3. Compute the scatter matrix $\mathbf{S} = \sum_{i=1}^n \mathbf{z}_i \mathbf{z}_i^t$
4. Compute eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_k$ corresponding to the k largest eigenvalues of \mathbf{S}
5. Let $\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_k$ be the columns of matrix $\mathbf{E} = [\mathbf{e}_1 \cdots \mathbf{e}_k]$
6. The desired \mathbf{y} which is the closest approximation to \mathbf{x} is $\mathbf{y} = \mathbf{E}^t \mathbf{z}$

PCA Example Using Matlab

- Let $D = \{(1,2), (2,3), (3,2), (4,4), (5,4), (6,7), (7,6), (9,7)\}$
- Convenient to arrange data in array

$$X = \begin{bmatrix} 1 & 2 \\ \vdots & \vdots \\ 9 & 7 \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_8 \end{bmatrix}$$

- Mean $\mu = \text{mean}(x) = [4.6 \ 4.4]$



- Subtract mean from data to get new data array Z

$$Z = X - \begin{bmatrix} \mu \\ \vdots \\ \mu \end{bmatrix} = X - \text{repmat}(\mu, 8, 1) = \begin{bmatrix} -3.6 & -4.4 \\ \vdots & \vdots \\ 4.4 & 2.6 \end{bmatrix}$$

- Compute the scatter matrix S

$$S = 7 * \text{cov}(Z) = [-3.6 \ -4.4] \begin{bmatrix} -3.6 \\ -4.4 \end{bmatrix} + \dots + [4.4 \ 2.6] \begin{bmatrix} 4.4 \\ 2.6 \end{bmatrix} = \begin{bmatrix} 57 & 40 \\ 40 & 34 \end{bmatrix}$$

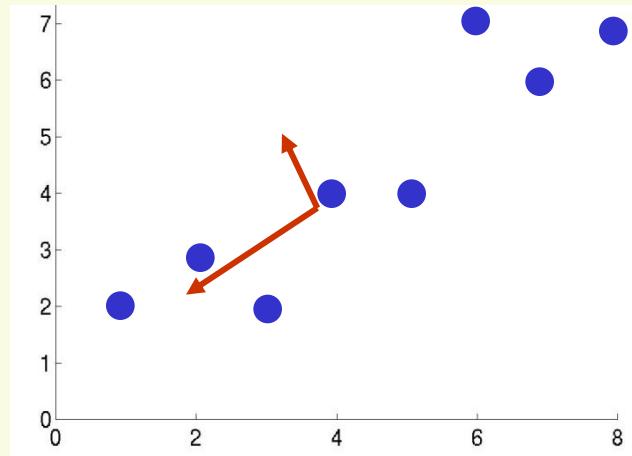
matlab uses unbiased estimate for covariance, so $S=(n-1)\text{cov}(Z)$*

PCA Example Using Matlab

- Use $[V, D] = \text{eig}(S)$ to get eigenvalues and eigenvectors of S

$$\lambda_1 = 87 \text{ and } e_1 = \begin{bmatrix} -0.8 \\ -0.6 \end{bmatrix}$$

$$\lambda_2 = 3.8 \text{ and } e_2 = \begin{bmatrix} 0.6 \\ -0.8 \end{bmatrix}$$



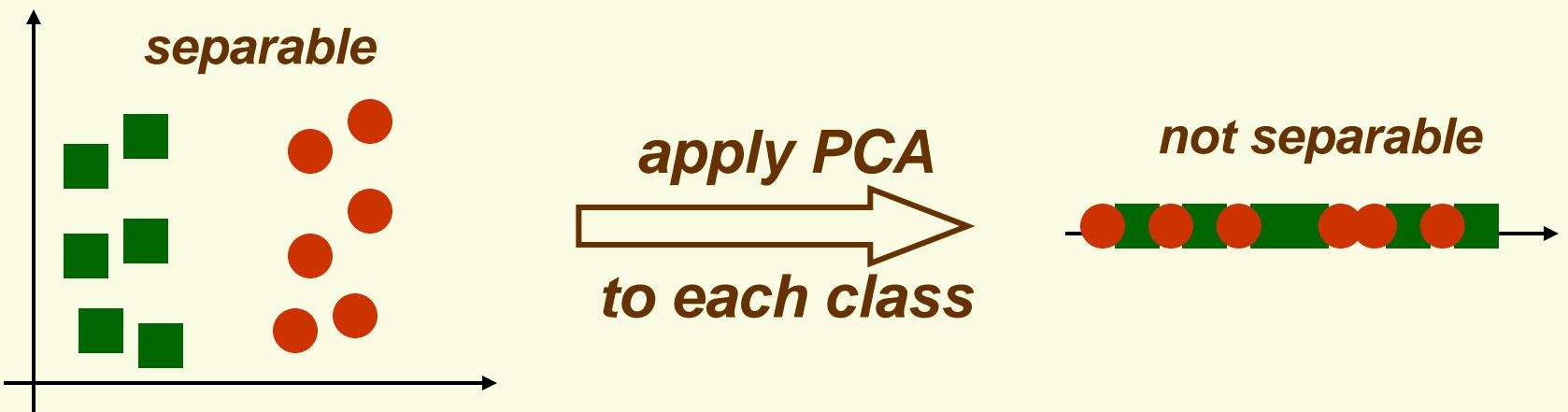
- Projection to 1D space in the direction of e_1 ,

$$Y = e_1^t Z^t = \left([-0.8 \ -0.6] \begin{bmatrix} -3.6 & \dots & 4.4 \\ -4.4 & \dots & 2.6 \end{bmatrix} \right) = [4.3 \ \dots \ -5.1] \\ = [y_1 \ \dots \ y_8]$$

Fisher LDA
MDA

Data Representation vs. Data Classification

- PCA finds the most accurate *data representation* in a lower dimensional space
 - Project data in the directions of maximum variance
- However the directions of maximum variance may be useless for classification

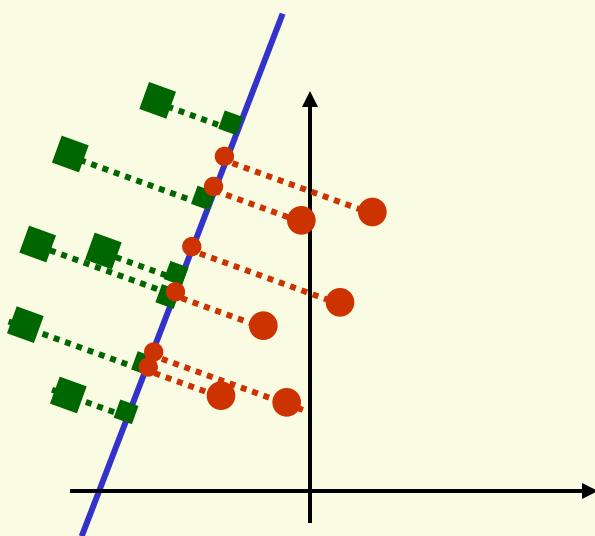


- Fisher Linear Discriminant projects to a line which preserves direction useful for *data classification*

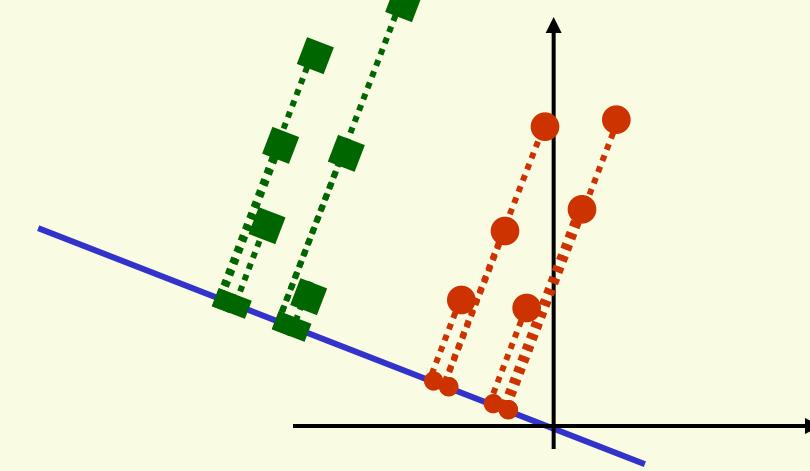
Fisher Linear Discriminant

- Main idea: find projection to a line s.t. samples from different classes are well separated

Example in 2D



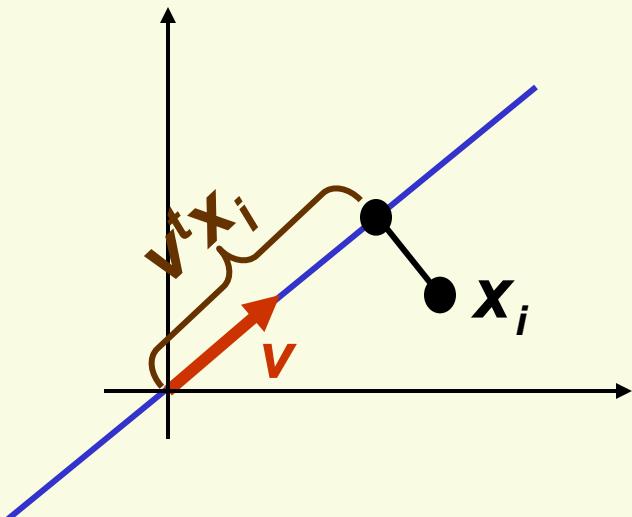
*bad line to project to,
classes are mixed up*



*good line to project to,
classes are well separated*

Fisher Linear Discriminant

- Suppose we have 2 classes and d -dimensional samples $\mathbf{x}_1, \dots, \mathbf{x}_n$ where
 - n_1 samples come from the first class
 - n_2 samples come from the second class
- consider projection on a line
- Let the line direction be given by unit vector \mathbf{v}



- Thus the projection of sample \mathbf{x}_i onto a line in direction \mathbf{v} is given by $\mathbf{v}^t \mathbf{x}_i$

Fisher Linear Discriminant

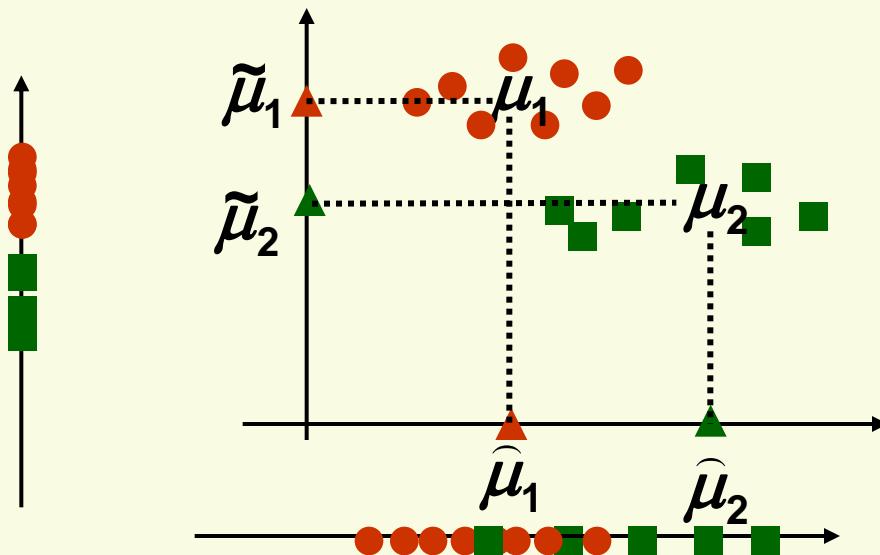
- How to measure separation between projections of different classes?
- Let $\tilde{\mu}_1$ and $\tilde{\mu}_2$ be the means of projections of classes 1 and 2
- Let μ_1 and μ_2 be the means of classes 1 and 2
- $|\tilde{\mu}_1 - \tilde{\mu}_2|$ seems like a good measure

$$\tilde{\mu}_1 = \frac{1}{n_1} \sum_{x_i \in C1}^{n_1} v^t x_i = v^t \left(\frac{1}{n_1} \sum_{x_i \in C1}^{n_1} x_i \right) = v^t \mu_1$$

similarly, $\tilde{\mu}_2 = v^t \mu_2$

Fisher Linear Discriminant

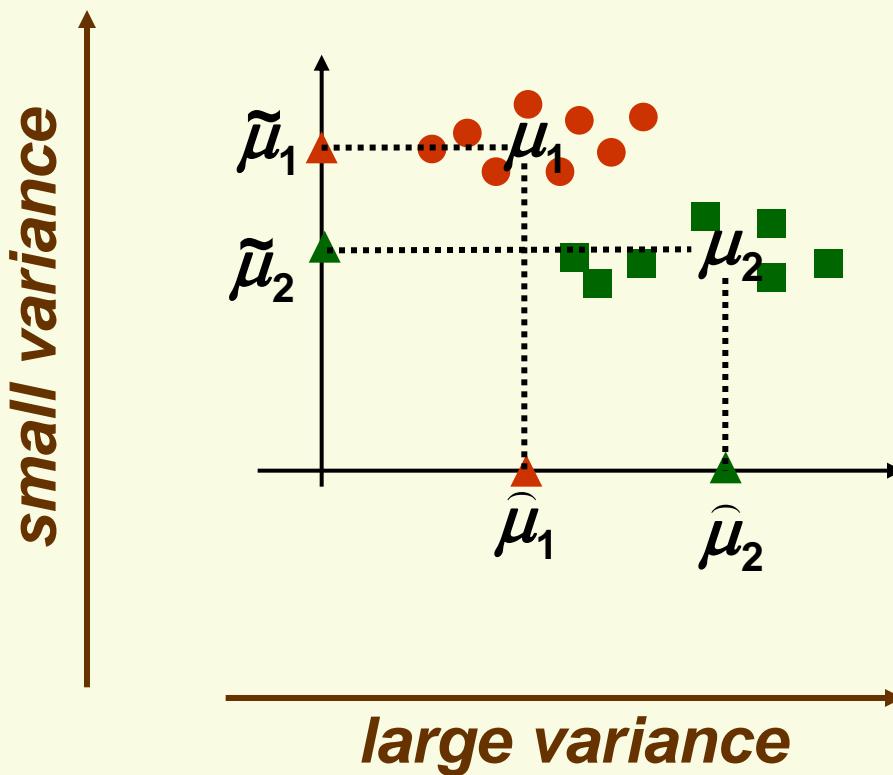
- How good is $|\tilde{\mu}_1 - \tilde{\mu}_2|$ as a measure of separation?
 - The larger $|\tilde{\mu}_1 - \tilde{\mu}_2|$, the better is the expected separation



- the vertical axes is a better line than the horizontal axes to project to for class separability
- however $|\hat{\mu}_1 - \hat{\mu}_2| > |\tilde{\mu}_1 - \tilde{\mu}_2|$

Fisher Linear Discriminant

- The problem with $|\tilde{\mu}_1 - \tilde{\mu}_2|$ is that it does not consider the variance of the classes



Fisher Linear Discriminant

- We need to normalize $|\tilde{\mu}_1 - \tilde{\mu}_2|$ by a factor which is proportional to variance
- 1D samples $\mathbf{z}_1, \dots, \mathbf{z}_n$. Sample mean is $\mu_z = \frac{1}{n} \sum_{i=1}^n \mathbf{z}_i$

- Define their **scatter** as

$$s = \sum_{i=1}^n (\mathbf{z}_i - \mu_z)^2$$

- Thus scatter is just sample variance multiplied by n
 - scatter measures the same thing as variance, the spread of data around the mean
 - scatter is just on different scale than variance



Fisher Linear Discriminant

- Fisher Solution: normalize $|\tilde{\mu}_1 - \tilde{\mu}_2|$ by scatter
- Let $y_i = v^t x_i$, i.e. y_i 's are the projected samples
- Scatter for projected samples of class 1 is

$$\tilde{s}_1^2 = \sum_{y_i \in \text{Class 1}} (y_i - \tilde{\mu}_1)^2$$

- Scatter for projected samples of class 2 is

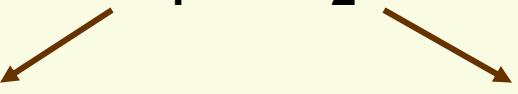
$$\tilde{s}_2^2 = \sum_{y_i \in \text{Class 2}} (y_i - \tilde{\mu}_2)^2$$

Fisher Linear Discriminant

- We need to normalize by both scatter of class 1 and scatter of class 2
- Thus Fisher linear discriminant is to project on line in the direction \mathbf{v} which maximizes

want projected means are far from each other

$$J(\mathbf{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$



want scatter in class 1 to be as small as possible, i.e. samples of class 1 cluster around the projected mean $\tilde{\mu}_1$

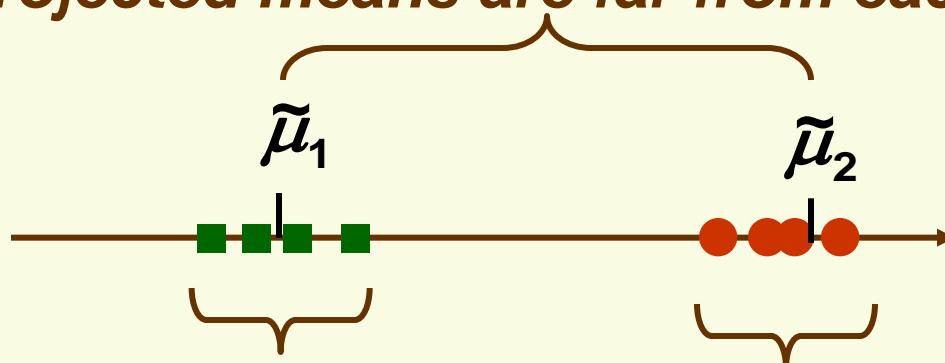
want scatter in class 2 to be as small as possible, i.e. samples of class 2 cluster around the projected mean $\tilde{\mu}_2$

Fisher Linear Discriminant

$$J(\mathbf{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

- If we find \mathbf{v} which makes $J(\mathbf{v})$ large, we are guaranteed that the classes are well separated

projected means are far from each other



small \tilde{s}_1 implies that projected samples of class 1 are clustered around projected mean

small \tilde{s}_2 implies that projected samples of class 2 are clustered around projected mean

Fisher Linear Discriminant Derivation

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{S}_1^2 + \tilde{S}_2^2}$$

- All we need to do now is to express J explicitly as a function of v and maximize it
 - straightforward but need linear algebra and Calculus (the derivation is shown in the next few slides.)
 - The solution is found by **generalized eigenvalue problem** $\Rightarrow S_B v = \lambda S_W v$

between class scatter matrix $S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^t$

within the class scatter matrix $S_W = S_1 + S_2$

$$S_1 = \sum_{x_i \in Class1} (x_i - \mu_1)(x_i - \mu_1)^t \quad S_2 = \sum_{x_i \in Class2} (x_i - \mu_2)(x_i - \mu_2)^t$$

Fisher Linear Discriminant Derivation

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

- Define the separate class scatter matrices S_1 and S_2 for classes 1 and 2. These measure the scatter of original samples x_i (before projection)

$$S_1 = \sum_{x_i \in \text{Class1}} (x_i - \mu_1)(x_i - \mu_1)^t$$

$$S_2 = \sum_{x_i \in \text{Class2}} (x_i - \mu_2)(x_i - \mu_2)^t$$

Fisher Linear Discriminant Derivation

- Now define the **within** the class scatter matrix

$$\mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2$$

- Recall that $\tilde{\mathbf{s}}_1^2 = \sum_{y_i \in \text{Class 1}} (y_i - \tilde{\mu}_1)^2$

- Using $y_i = \mathbf{v}^t \mathbf{x}_i$ and $\tilde{\mu}_1 = \mathbf{v}^t \mu_1$

$$\begin{aligned}\tilde{\mathbf{s}}_1^2 &= \sum_{y_i \in \text{Class 1}} (\mathbf{v}^t \mathbf{x}_i - \mathbf{v}^t \mu_1)^2 \\ &= \sum_{y_i \in \text{Class 1}} (\mathbf{v}^t (\mathbf{x}_i - \mu_1))^t (\mathbf{v}^t (\mathbf{x}_i - \mu_1)) \\ &= \sum_{y_i \in \text{Class 1}} ((\mathbf{x}_i - \mu_1)^t \mathbf{v})^t ((\mathbf{x}_i - \mu_1)^t \mathbf{v}) \\ &= \sum_{y_i \in \text{Class 1}} \mathbf{v}^t (\mathbf{x}_i - \mu_1) (\mathbf{x}_i - \mu_1)^t \mathbf{v} = \mathbf{v}^t \mathbf{S}_1 \mathbf{v}\end{aligned}$$

Fisher Linear Discriminant Derivation

- Similarly $\tilde{\mathbf{S}}_2^2 = \mathbf{v}^t \mathbf{S}_2 \mathbf{v}$
- Therefore $\tilde{\mathbf{S}}_1^2 + \tilde{\mathbf{S}}_2^2 = \mathbf{v}^t \mathbf{S}_1 \mathbf{v} + \mathbf{v}^t \mathbf{S}_2 \mathbf{v} = \mathbf{v}^t \mathbf{S}_w \mathbf{v}$
- Define between class scatter matrix

$$\mathbf{S}_B = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^t$$

- \mathbf{S}_B measures separation between the means of two classes (before projection)
- Let's rewrite the separations of the projected means

$$\begin{aligned}(\tilde{\boldsymbol{\mu}}_1 - \tilde{\boldsymbol{\mu}}_2)^2 &= (\mathbf{v}^t \boldsymbol{\mu}_1 - \mathbf{v}^t \boldsymbol{\mu}_2)^2 \\&= \mathbf{v}^t (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^t \mathbf{v} \\&= \mathbf{v}^t \mathbf{S}_B \mathbf{v}\end{aligned}$$

Fisher Linear Discriminant Derivation

- Thus our objective function can be written:

$$J(\mathbf{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{\mathbf{v}^t \mathbf{S}_B \mathbf{v}}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}}$$

- Maximize $J(\mathbf{v})$ by taking the derivative w.r.t. \mathbf{v} and setting it to 0

$$\begin{aligned}\frac{d}{d\mathbf{v}} J(\mathbf{v}) &= \frac{\left(\frac{d}{d\mathbf{v}} \mathbf{v}^t \mathbf{S}_B \mathbf{v} \right) \mathbf{v}^t \mathbf{S}_W \mathbf{v} - \left(\frac{d}{d\mathbf{v}} \mathbf{v}^t \mathbf{S}_W \mathbf{v} \right) \mathbf{v}^t \mathbf{S}_B \mathbf{v}}{(\mathbf{v}^t \mathbf{S}_W \mathbf{v})^2} \\ &= \frac{(2\mathbf{S}_B \mathbf{v}) \mathbf{v}^t \mathbf{S}_W \mathbf{v} - (2\mathbf{S}_W \mathbf{v}) \mathbf{v}^t \mathbf{S}_B \mathbf{v}}{(\mathbf{v}^t \mathbf{S}_W \mathbf{v})^2} = 0\end{aligned}$$

Fisher Linear Discriminant Derivation

- Need to solve $\mathbf{v}^t \mathbf{S}_W \mathbf{v} (\mathbf{S}_B \mathbf{v}) - \mathbf{v}^t \mathbf{S}_B \mathbf{v} (\mathbf{S}_W \mathbf{v}) = 0$

$$\Rightarrow \frac{\mathbf{v}^t \mathbf{S}_W \mathbf{v} (\mathbf{S}_B \mathbf{v})}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}} - \frac{\mathbf{v}^t \mathbf{S}_B \mathbf{v} (\mathbf{S}_W \mathbf{v})}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}} = 0$$

$$\Rightarrow \mathbf{S}_B \mathbf{v} - \frac{\mathbf{v}^t \mathbf{S}_B \mathbf{v} (\mathbf{S}_W \mathbf{v})}{\mathbf{v}^t \mathbf{S}_W \mathbf{v}} = 0$$

$\boxed{\mathbf{v}^t \mathbf{S}_B \mathbf{v} (\mathbf{S}_W \mathbf{v}) / \mathbf{v}^t \mathbf{S}_W \mathbf{v}} = \lambda$

$$\Rightarrow \underbrace{\mathbf{S}_B \mathbf{v}}_{\lambda \mathbf{S}_W \mathbf{v}} = \lambda \mathbf{S}_W \mathbf{v}$$

generalized eigenvalue problem

Fisher Linear Discriminant Derivation

$$\mathbf{S}_B \mathbf{v} = \lambda \mathbf{S}_W \mathbf{v}$$

- If \mathbf{S}_W has full rank (the inverse exists), can convert this to a standard eigenvalue problem

$$\mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{v} = \lambda \mathbf{v}$$

- Turn's out that we don't have to solve for eigenvalues; the solution is:

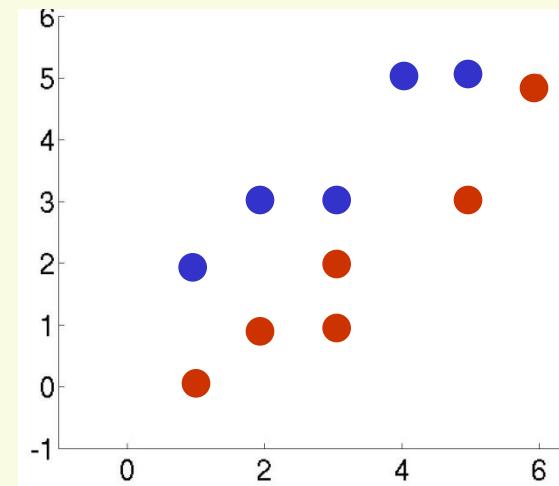
$$\boxed{\mathbf{v} = \mathbf{S}_W^{-1}(\mu_1 - \mu_2)}$$

Fisher Linear Discriminant Example

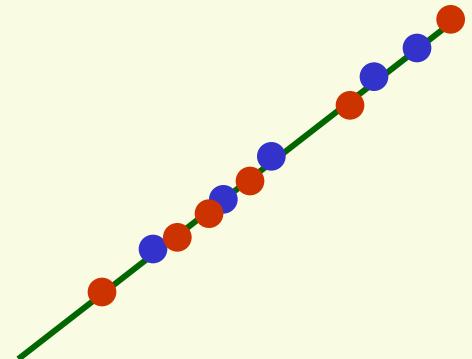
- Data
 - Class 1 has 5 samples $\mathbf{c}_1 = [(1,2), (2,3), (3,3), (4,5), (5,5)]$
 - Class 2 has 6 samples $\mathbf{c}_2 = [(1,0), (2,1), (3,1), (3,2), (5,3), (6,5)]$
- Arrange data in 2 separate matrices

$$\mathbf{c}_1 = \begin{bmatrix} 1 & 2 \\ \vdots & \vdots \\ 5 & 5 \end{bmatrix}$$

$$\mathbf{c}_2 = \begin{bmatrix} 1 & 0 \\ \vdots & \vdots \\ 6 & 5 \end{bmatrix}$$



- Notice that PCA performs very poorly on this data because the direction of largest variance is not helpful for classification



Fisher Linear Discriminant Example

- First compute the mean for each class

$$\mu_1 = \text{mean}(c_1) = [3 \ 3.6] \quad \mu_2 = \text{mean}(c_2) = [3.3 \ 2]$$

- Compute scatter matrices S_1 and S_2 for each class

$$S_1 = 4 * \text{cov}(c_1) = \begin{bmatrix} 10 & 8.0 \\ 8.0 & 7.2 \end{bmatrix} \quad S_2 = 5 * \text{cov}(c_2) = \begin{bmatrix} 17.3 & 16 \\ 16 & 16 \end{bmatrix}$$

- Within the class scatter:

$$S_w = S_1 + S_2 = \begin{bmatrix} 27.3 & 24 \\ 24 & 23.2 \end{bmatrix}$$

- it has full rank, don't have to solve for eigenvalues

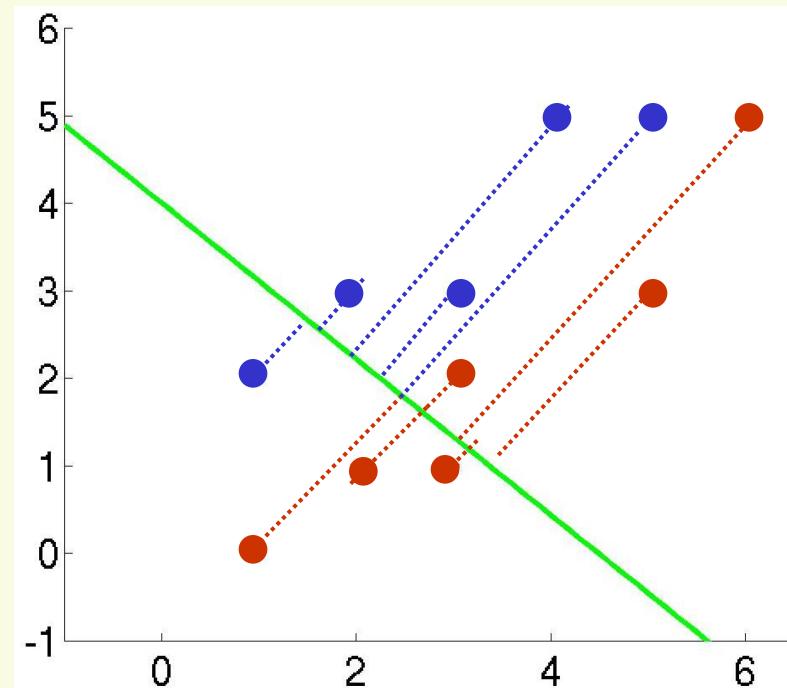
- The inverse of S_w is $S_w^{-1} = \text{inv}(S_w) = \begin{bmatrix} 0.39 & -0.41 \\ -0.41 & 0.47 \end{bmatrix}$

- Finally, the optimal line direction v

$$v = S_w^{-1}(\mu_1 - \mu_2) = \begin{bmatrix} -0.79 \\ 0.89 \end{bmatrix}$$

Fisher Linear Discriminant Example

- Notice, as long as the line has the right direction, its exact position does not matter
- Last step is to compute the actual ***1D*** vector ***y***. Let's do it separately for each class

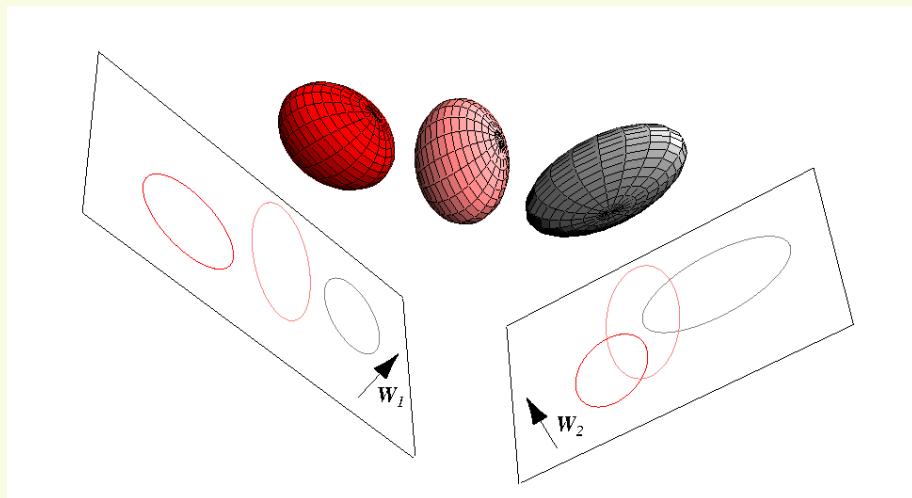


$$Y_1 = \mathbf{v}^t \mathbf{c}_1^t = [-0.79 \quad 0.89] \begin{bmatrix} 1 \\ 2 \\ \vdots \\ 5 \end{bmatrix} = [0.98 \quad \dots \quad 0.48]$$

$$Y_2 = \mathbf{v}^t \mathbf{c}_2^t = [-0.79 \quad 0.89] \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 5 \end{bmatrix} = [-0.79 \quad \dots \quad -0.31]$$

Multiple Discriminant Analysis (MDA)

- Can generalize FLD to multiple classes
- In case of **c** classes, can reduce dimensionality to 1, 2, 3,..., **c**-1 dimensions
- Project sample x_i to a linear subspace $y_i = V^t x_i$
 - V is called projection matrix



Multiple Discriminant Analysis (MDA)

- Let
 - n_i by the number of samples of class i
 - μ_i be the sample mean of class i
 - μ be the total mean of all samples

$$\mu_i = \frac{1}{n_i} \sum_{x \in \text{class } i} x \quad \mu = \frac{1}{n} \sum_{x_i} x_i$$

- Objective function: $J(V) = \frac{\det(V^t S_B V)}{\det(V^t S_W V)}$

- within the class scatter matrix S_W is

$$S_W = \sum_{i=1}^c S_i = \sum_{i=1}^c \sum_{x_k \in \text{class } i} (x_k - \mu_i)(x_k - \mu_i)^t$$

- between the class scatter matrix S_B is

$$S_B = \sum_{i=1}^c n_i (\mu_i - \mu)(\mu_i - \mu)^t$$

maximum rank is $c - 1$

Multiple Discriminant Analysis (MDA)

- Objective function:

$$J(\mathbf{V}) = \frac{\det(\mathbf{V}^t \mathbf{S}_B \mathbf{V})}{\det(\mathbf{V}^t \mathbf{S}_W \mathbf{V})}$$

- It can be shown that “scatter” of the samples is directly proportional to the determinant of the scatter matrix
 - the larger $\det(S)$, the more scattered samples are
 - $\det(S)$ is the product of eigenvalues of S
- Thus we are seeking transformation \mathbf{V} which maximizes the between class scatter and minimizes the within-class scatter

Multiple Discriminant Analysis (MDA)

$$J(\mathbf{v}) = \frac{\det(\mathbf{v}^t \mathbf{S}_B \mathbf{v})}{\det(\mathbf{v}^t \mathbf{S}_W \mathbf{v})}$$

- First solve the **generalized eigenvalue** problem:

$$\mathbf{S}_B \mathbf{v} = \lambda \mathbf{S}_W \mathbf{v}$$

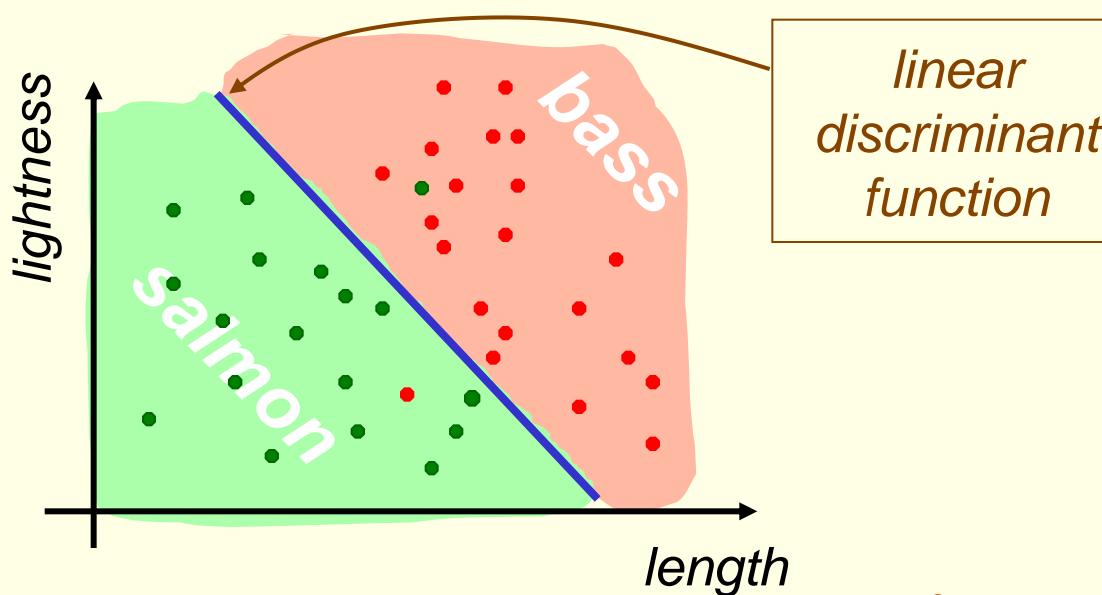
- At most **c-1** eigenvalues are nonzero.
- Let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{c-1}$ be the corresponding eigenvectors
- The optimal projection matrix \mathbf{V} to a subspace of dimension **k** is given by the eigenvectors corresponding to the largest **k** eigenvalues
- Thus can project to a subspace of dimension at most **c-1**

Linear Discriminant Functions

Linear discriminant functions on Road Map

- No probability distribution (no shape or parameters are known)
- Labeled data
- The shape of discriminant functions is known

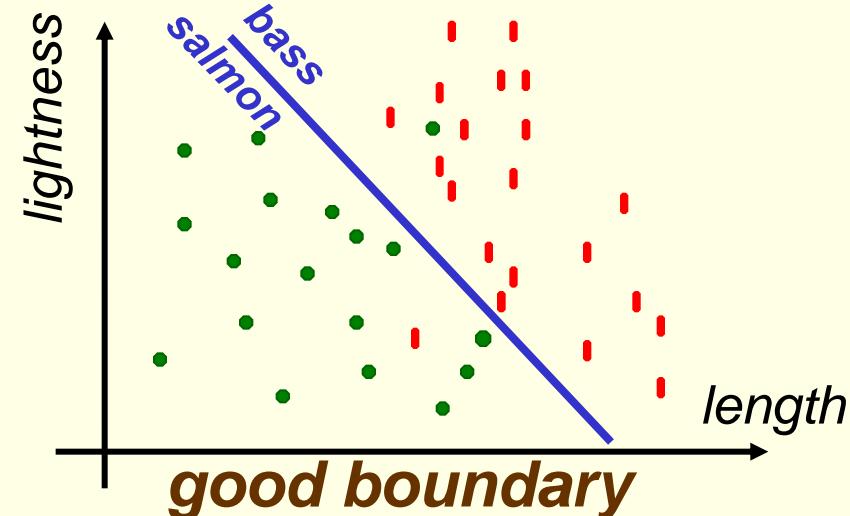
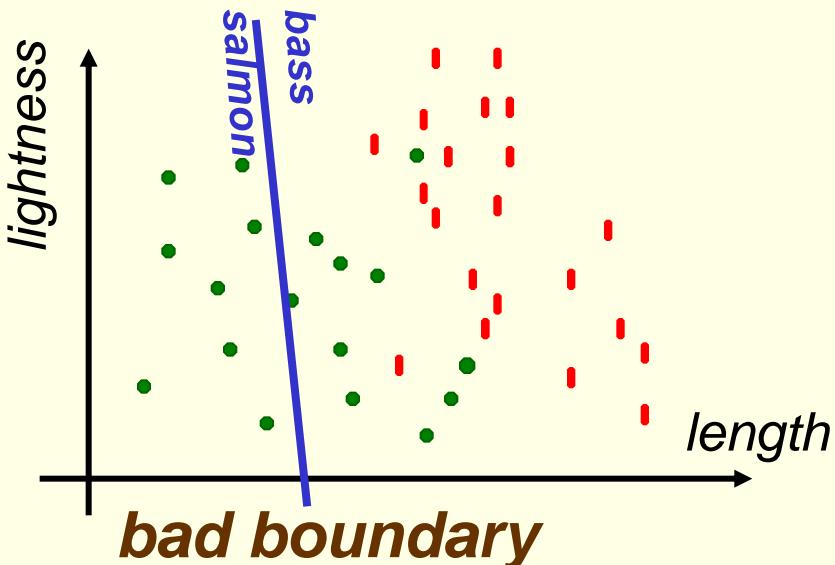
a lot is known



- Need to estimate parameters of the discriminant function (parameters of the line in case of linear discriminant)

little is known

Linear Discriminant Functions: Basic Idea



- Have samples from 2 classes x_1, x_2, \dots, x_n
- Assume 2 classes can be separated by a linear boundary $I(\theta)$ with some unknown parameters θ
- Fit the “best” boundary to data by optimizing over parameters θ . **How?**
- Minimize a criterion function.
 - Obvious choice: Minimize classification error on training data. **(Does not guarantee small test error)**

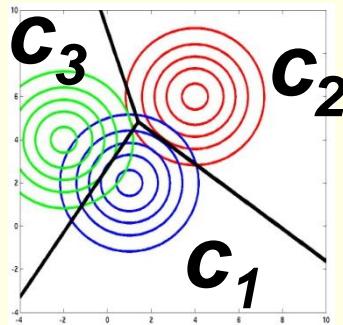
Parametric Methods vs.

Discriminant Functions

Assume the shape of density for classes is known $p_1(\mathbf{x}|\theta_1)$,
 $p_2(\mathbf{x}|\theta_2), \dots$

Estimate $\theta_1, \theta_2, \dots$ from data

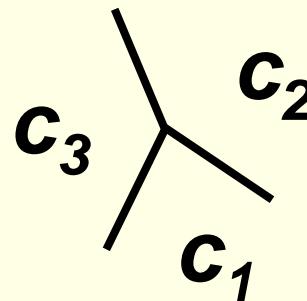
Use a Bayesian classifier to find decision regions



Assume discriminant functions are of known shape $I(\theta_1), I(\theta_2), \dots$, with parameters $\theta_1, \theta_2, \dots$

Estimate $\theta_1, \theta_2, \dots$ from data

Use discriminant functions for classification



Parametric Methods vs. Discriminant Functions

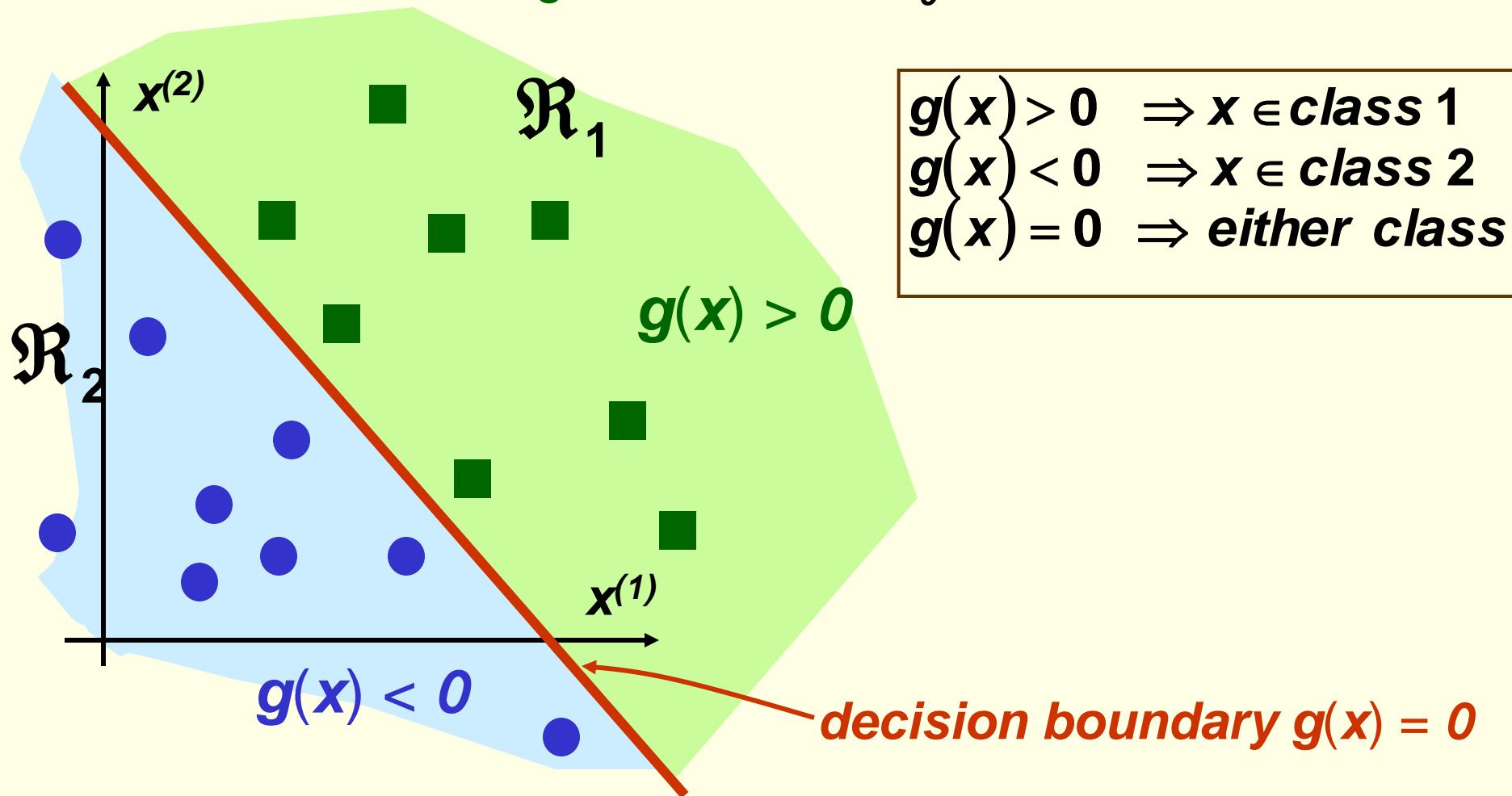
- In theory, Bayesian classifier minimizes the risk, but in practice:
 - do not have confidence in assumed model shapes;
 - do not really need the actual density functions in the end.
- Estimating accurate density functions is much harder than estimating accurate discriminant functions
 - Some argue that estimating densities should be skipped. Why solve a harder problem than needed ?

LDF: Introduction

- Discriminant functions can be more general than linear.
- For now, we will study linear discriminant functions
 - Simple model (should try simpler models first)
 - Analytically tractable.
- Linear Discriminant functions are optimal for Gaussian distributions with equal covariance.
- May not be optimal for other data distributions, but they are very simple to use.
- Knowledge of class densities is not required when using linear discriminant functions.
 - we can say that this is a non-parametric approach

LDF: 2 Classes

- A discriminant function is linear if it can be written as
$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$
- \mathbf{w} is called the weight vector and w_0 called bias or threshold

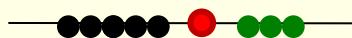


LDF: 2 Classes

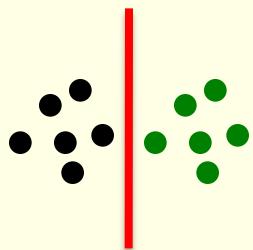
- Decision boundary $\mathbf{g}(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0 = 0$ is a **hyperplane**

- A hyperplane is

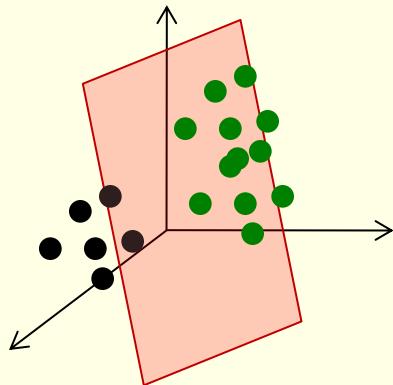
- a point in 1D



- a line in 2D



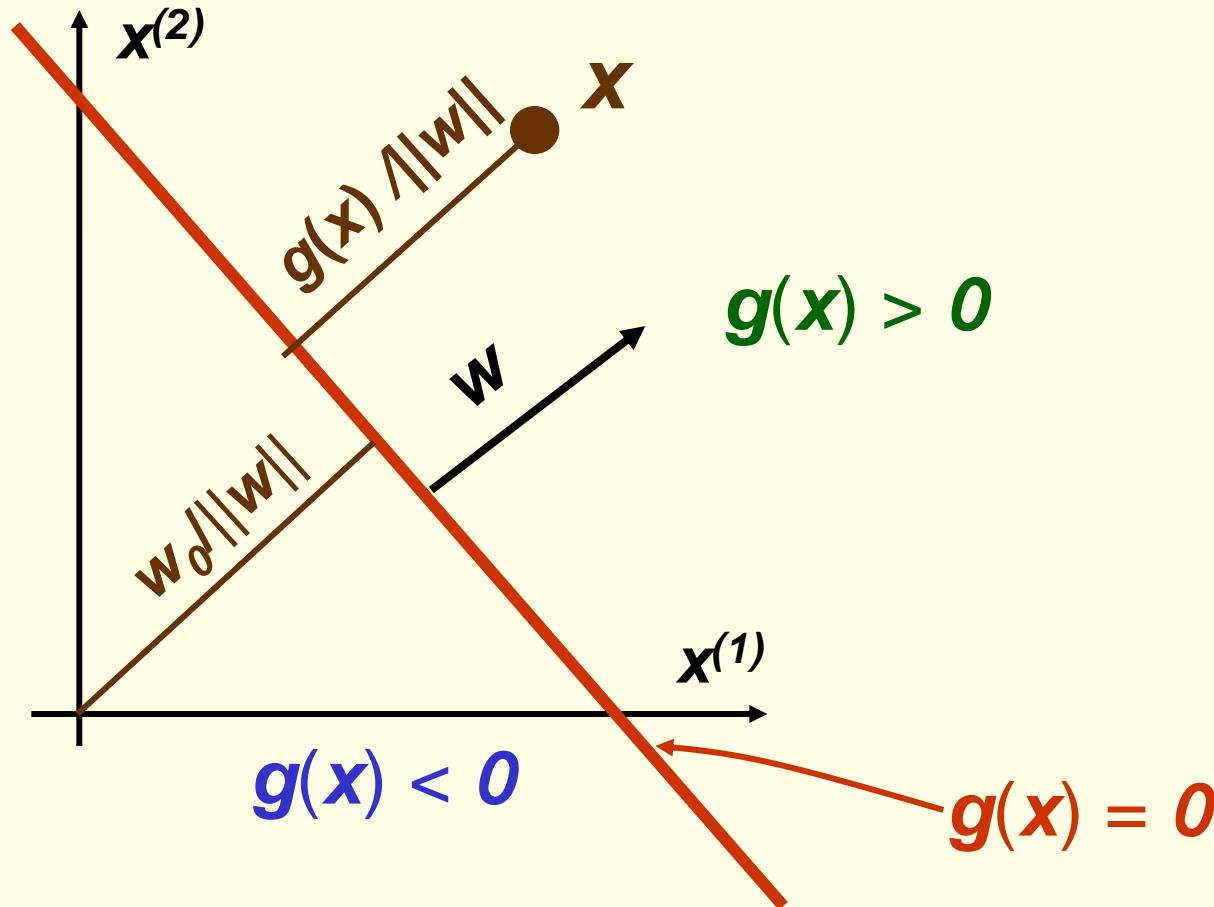
- a plane in 3D



LDF: 2 Classes

$$g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + w_0$$

- \mathbf{w} determines orientation of the decision hyperplane
- w_0 determines location of the decision surface



LDF: Many Classes

- Suppose we have m classes
- Define m linear discriminant functions

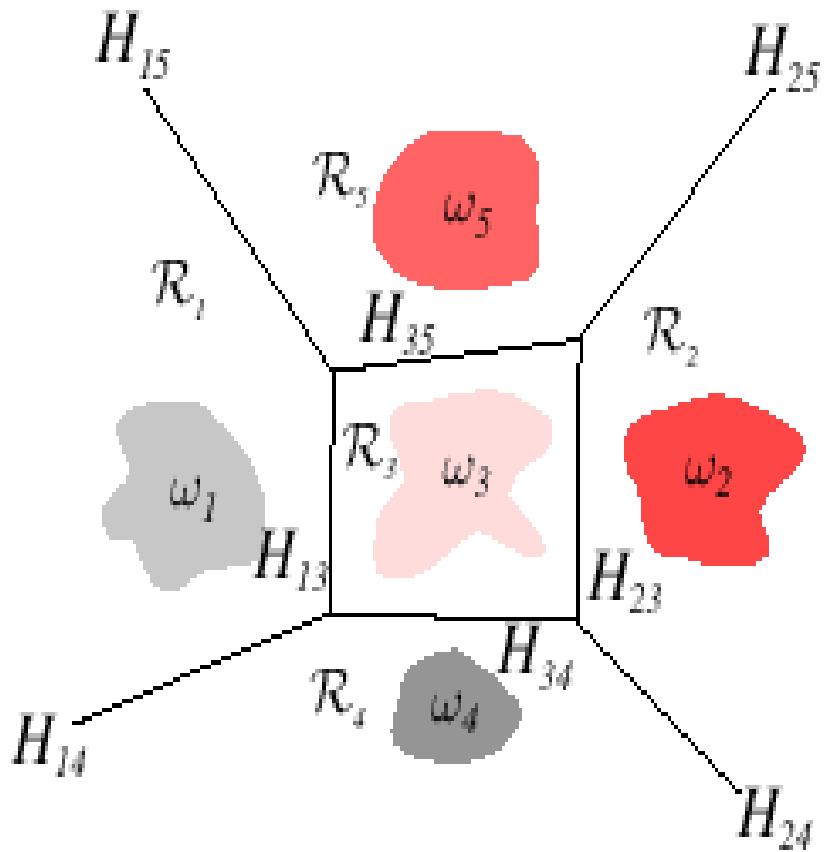
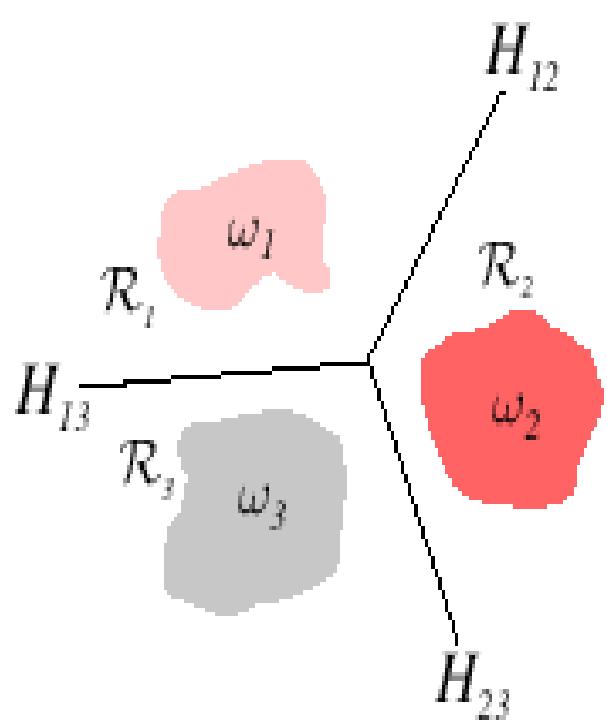
$$g_i(\mathbf{x}) = \mathbf{w}_i^t \mathbf{x} + w_{i0} \quad i = 1, \dots, m$$

- Given \mathbf{x} , assign class c_i if

$$g_i(\mathbf{x}) \geq g_j(\mathbf{x}) \quad \forall j \neq i$$

- Such classifier is called a *linear machine*
- A linear machine divides the feature space into c decision regions, with $g_i(\mathbf{x})$ being the largest discriminant if \mathbf{x} is in the region R_i

LDF: Many Classes



LDF: Many Classes

- For a two contiguous regions R_i and R_j ; the boundary that separates them is a portion of hyperplane H_{ij} defined by:

$$\begin{aligned} g_i(\mathbf{x}) = g_j(\mathbf{x}) &\Leftrightarrow \mathbf{w}_i^t \mathbf{x} + \mathbf{w}_{i0} = \mathbf{w}_j^t \mathbf{x} + \mathbf{w}_{j0} \\ &\Leftrightarrow (\mathbf{w}_i - \mathbf{w}_j)^t \mathbf{x} + (\mathbf{w}_{i0} - \mathbf{w}_{j0}) = 0 \end{aligned}$$

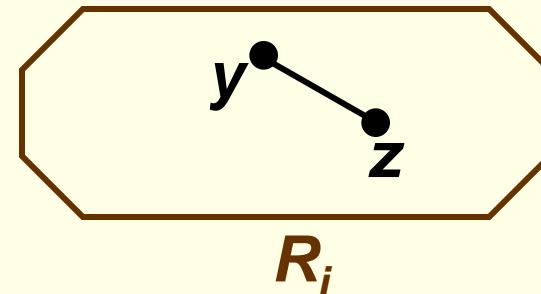
- Thus $\mathbf{w}_i - \mathbf{w}_j$ is normal to H_{ij}
- And distance from \mathbf{x} to H_{ij} is given by

$$d(\mathbf{x}, H_{ij}) = \frac{|g_i(\mathbf{x}) - g_j(\mathbf{x})|}{\|\mathbf{w}_i - \mathbf{w}_j\|}$$

LDF: Many Classes

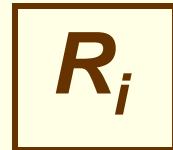
- Decision regions for a linear machine are **convex**

$$y, z \in R_i \Rightarrow \alpha y + (1 - \alpha)z \in R_i$$

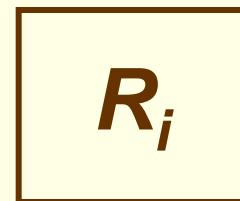


$$\begin{aligned} \forall j \neq i \quad g_i(y) &\geq g_j(y) \text{ and } g_i(z) \geq g_j(z) \Leftrightarrow \\ \Leftrightarrow \forall j \neq i \quad g_i(\alpha y + (1 - \alpha)z) &\geq g_j(\alpha y + (1 - \alpha)z) \end{aligned}$$

- In particular, decision regions must be spatially contiguous



*R_j is a valid
decision region*

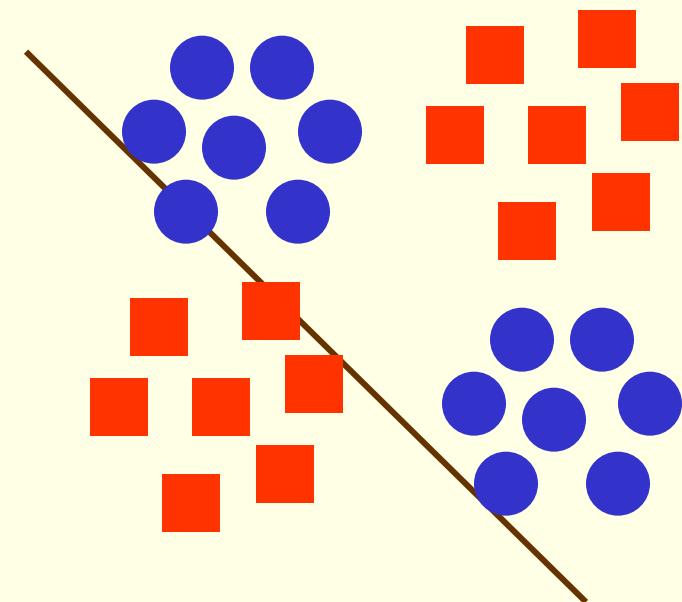


*R_j is not a valid
decision region*



LDF: Many Classes

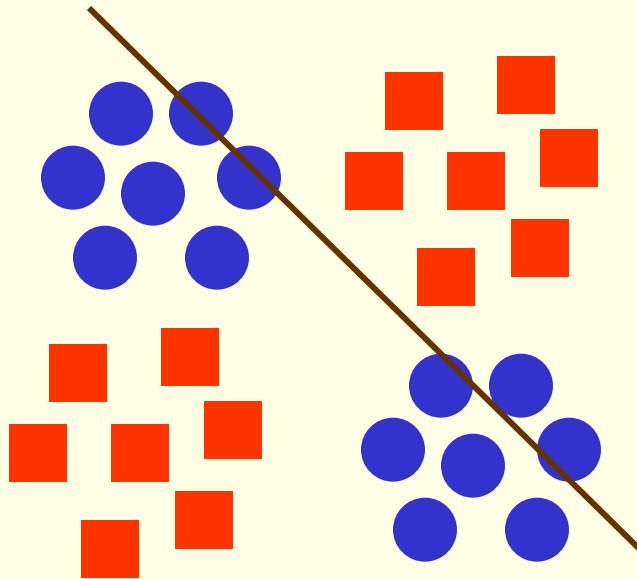
- Thus applicability of linear machine is mostly limited to unimodal conditional densities $p(x|\theta)$
 - even though we did not assume any parametric models
- Example:



LDF: Many Classes

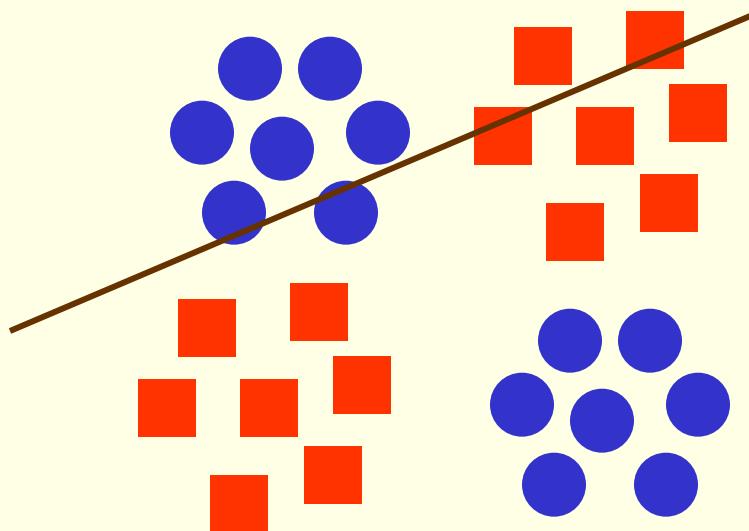
- Thus applicability of linear machine to mostly limited to unimodal conditional densities $p(x|\theta)$
 - even though we did not assume any parametric models

- Example:



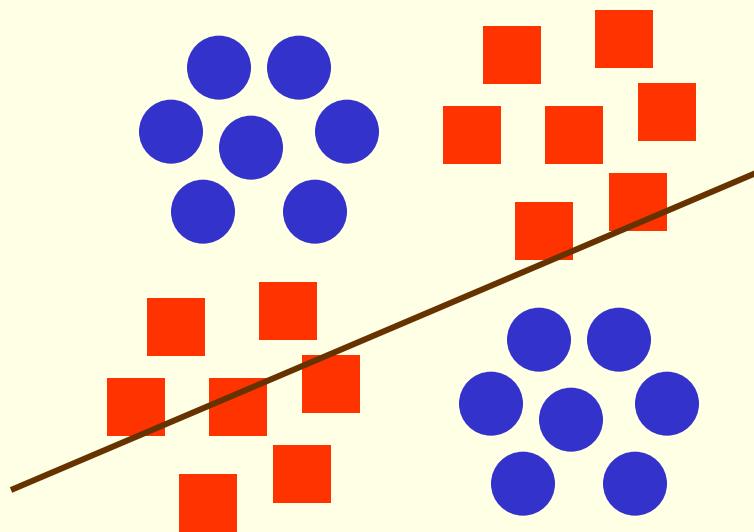
LDF: Many Classes

- Thus applicability of linear machine to mostly limited to unimodal conditional densities $p(x|\theta)$
 - even though we did not assume any parametric models
- Example:



LDF: Many Classes

- Thus applicability of linear machine to mostly limited to unimodal conditional densities $p(x|\theta)$
 - even though we did not assume any parametric models
- Example:



- need non-contiguous decision regions
- thus linear machine will fail

LDF: Augmented feature vector

- Linear discriminant function: $g(x) = w^t x + w_0$
- Can rewrite it: $g(x) = \underbrace{[w_0 \quad w^t]}_{\text{new weight vector } a} \underbrace{\begin{bmatrix} 1 \\ x \end{bmatrix}}_{\text{new feature vector } y} = a^t y = g(y)$
- y is called the **augmented feature vector**
- Added a dummy dimension to get a completely equivalent new **homogeneous** problem

old problem

$$g(x) = w^t x + w_0$$
$$\begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

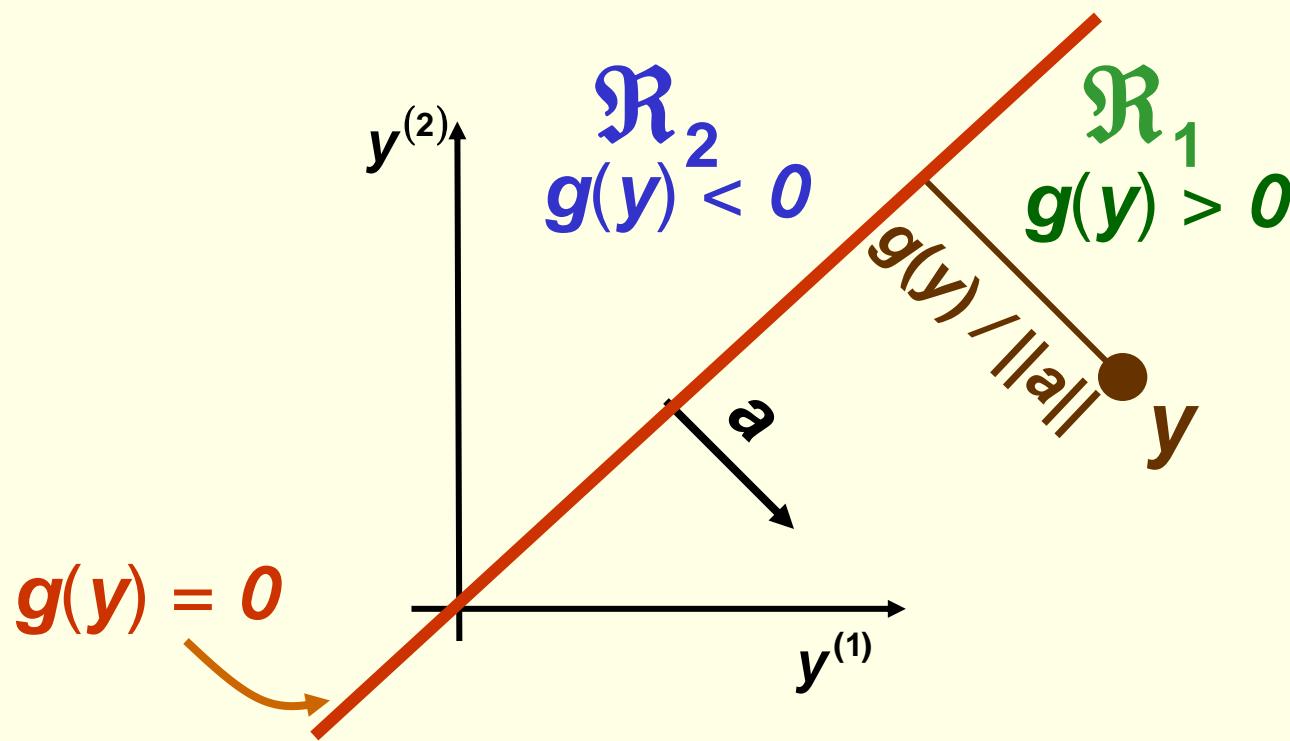
new problem

$$g(y) = a^t y$$
$$\begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$$

LDF: Augmented feature vector

- Feature augmenting is done for simpler notation
- From now on we always assume that we have augmented feature vectors
 - Given samples x_1, \dots, x_n convert them to augmented samples y_1, \dots, y_n by adding a new dimension of value 1

$$y_i = \begin{bmatrix} 1 \\ x_i \end{bmatrix}$$



LDF: Training Error

- For the rest of the lecture, assume we have 2 classes
- Samples $\mathbf{y}_1, \dots, \mathbf{y}_n$ some in class 1, some in class 2
- Use these samples to determine weights \mathbf{a} in the discriminant function $\mathbf{g}(\mathbf{y}) = \mathbf{a}^t \mathbf{y}$
- What should be our criterion for determining \mathbf{a} ?
 - For now, suppose we want to minimize the training error (that is the number of misclassified samples $\mathbf{y}_1, \dots, \mathbf{y}_n$)
- Recall that $\mathbf{g}(\mathbf{y}_i) > 0 \Rightarrow \mathbf{y}_i$ **classified** \mathbf{c}_1
 $\mathbf{g}(\mathbf{y}_i) < 0 \Rightarrow \mathbf{y}_i$ **classified** \mathbf{c}_2
- Thus training error is 0 if $\begin{cases} \mathbf{g}(\mathbf{y}_i) > 0 & \forall \mathbf{y}_i \in \mathbf{c}_1 \\ \mathbf{g}(\mathbf{y}_i) < 0 & \forall \mathbf{y}_i \in \mathbf{c}_2 \end{cases}$

LDF: Problem “Normalization”

- Thus training error is **0** if

$$\begin{cases} \mathbf{a}^t \mathbf{y}_i > 0 & \forall \mathbf{y}_i \in \mathcal{C}_1 \\ \mathbf{a}^t \mathbf{y}_i < 0 & \forall \mathbf{y}_i \in \mathcal{C}_2 \end{cases}$$

- Equivalently, training error is **0** if

$$\begin{cases} \mathbf{a}^t \mathbf{y}_i > 0 & \forall \mathbf{y}_i \in \mathcal{C}_1 \\ \mathbf{a}^t (-\mathbf{y}_i) > 0 & \forall \mathbf{y}_i \in \mathcal{C}_2 \end{cases}$$

- This suggests problem “normalization”:

1. Replace all examples from class \mathcal{C}_2 by their negative

$$\mathbf{y}_i \rightarrow -\mathbf{y}_i \quad \forall \mathbf{y}_i \in \mathcal{C}_2$$

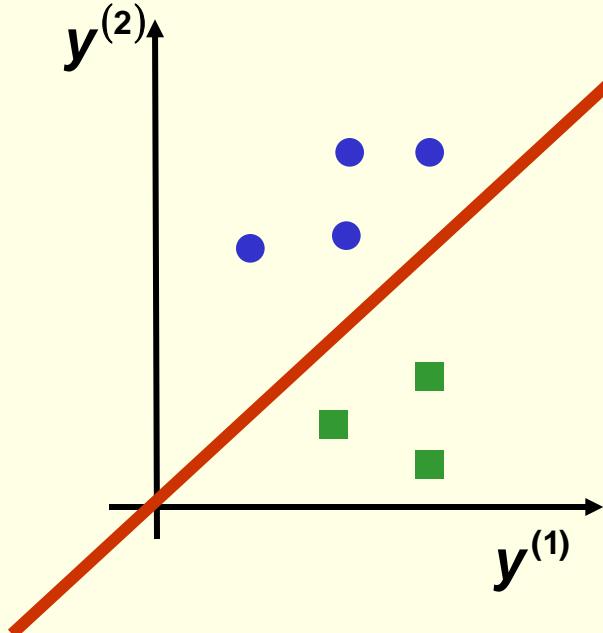
2. Seek weight vector \mathbf{a} s.t.

$$\mathbf{a}^t \mathbf{y}_i > 0 \quad \forall \mathbf{y}_i$$

- If such \mathbf{a} exists, it is called a *separating* or *solution* vector
- Original samples $\mathbf{x}_1, \dots, \mathbf{x}_n$ can indeed be separated by a line then

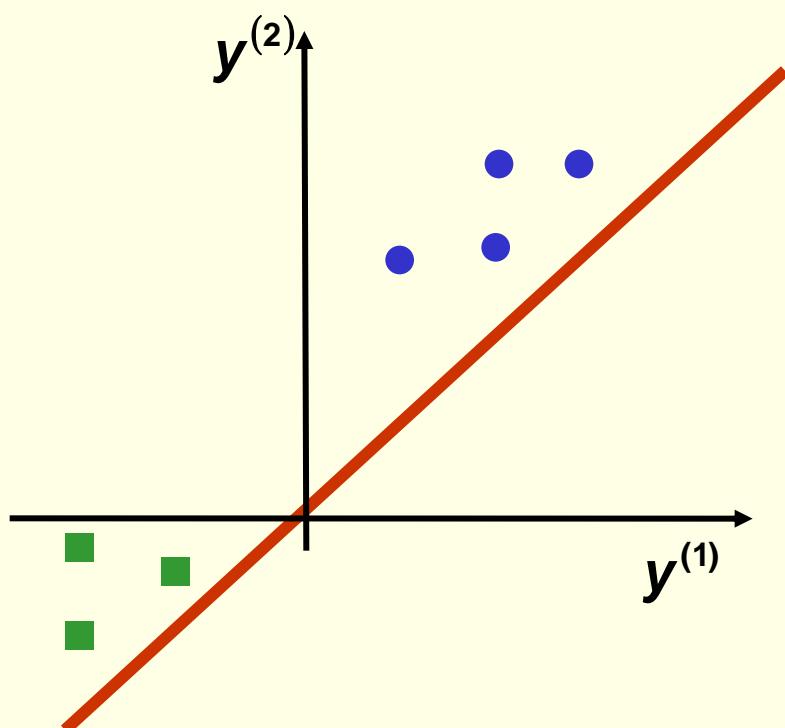
LDF: Problem “Normalization”

before normalization



Seek a hyperplane that
separates patterns from
different categories

after “normalization”

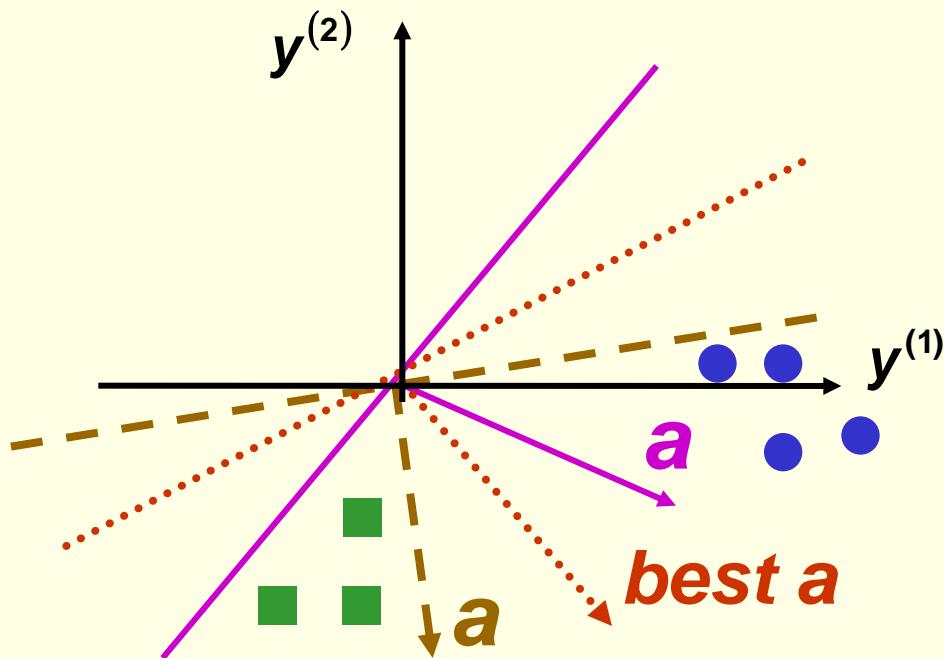


Seek hyperplane that
puts *normalized*
patterns on the same
(positive) side

LDF: Solution Region

- Find weight vector \mathbf{a} s.t. for all samples $\mathbf{y}_1, \dots, \mathbf{y}_n$

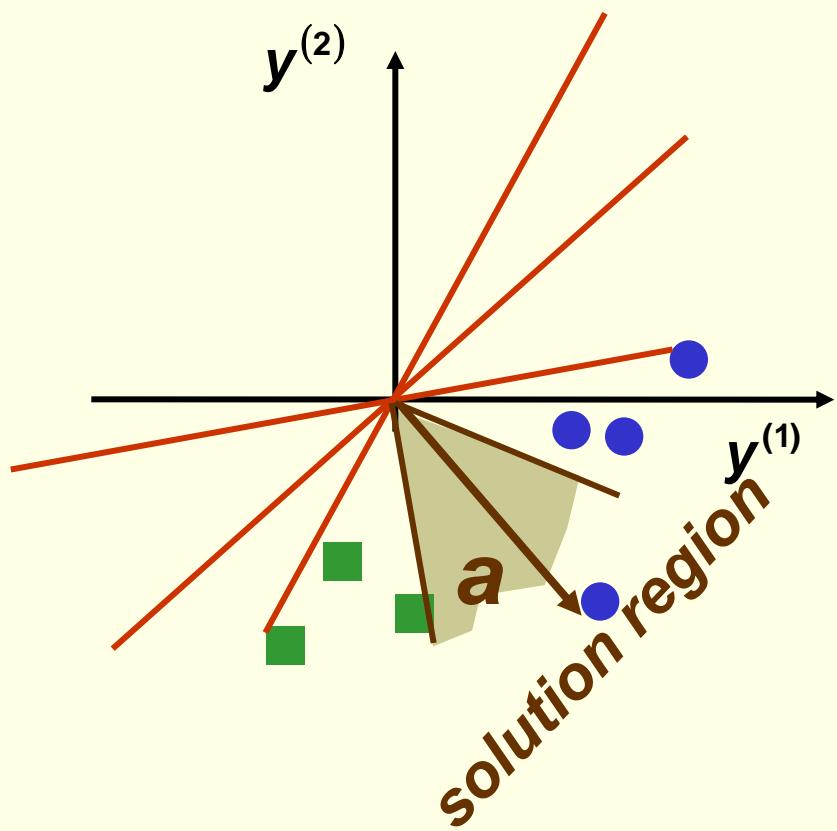
$$\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^d \mathbf{a}_k y_i^{(k)} > 0$$



- In general, there are many such solutions \mathbf{a}

LDF: Solution Region

- ***Solution region*** for \mathbf{a} : set of all possible solutions
 - defined in terms of normal \mathbf{a} to the separating hyperplane



Optimization

- Need to minimize a function of many variables

$$J(\mathbf{x}) = J(x_1, \dots, x_d)$$

- We know how to minimize $J(\mathbf{x})$

- Take partial derivatives and set them to zero

$$\begin{bmatrix} \frac{\partial}{\partial x_1} J(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_d} J(\mathbf{x}) \end{bmatrix} = \nabla J(\mathbf{x}) = \mathbf{0}$$

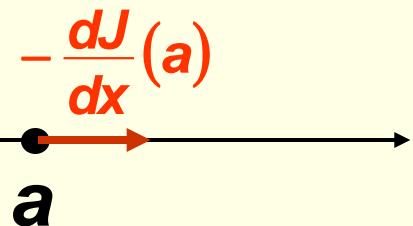
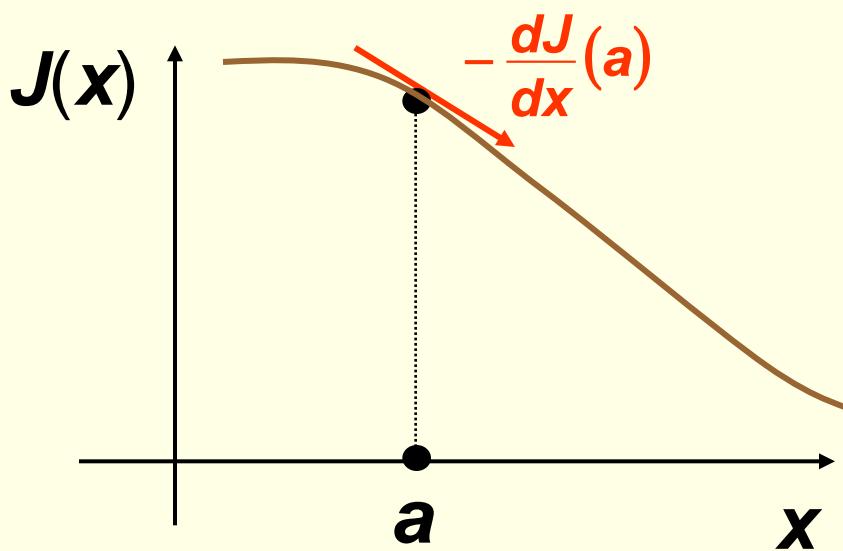
gradient

- However solving analytically is not always easy
- Sometimes it is not even possible to write down an analytical expression for the derivative, we will see an example later today.

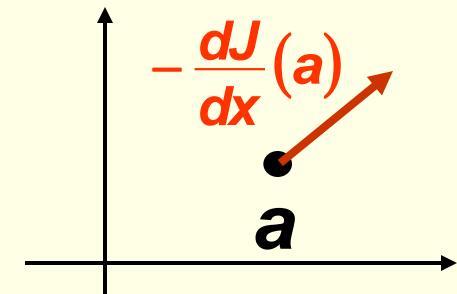
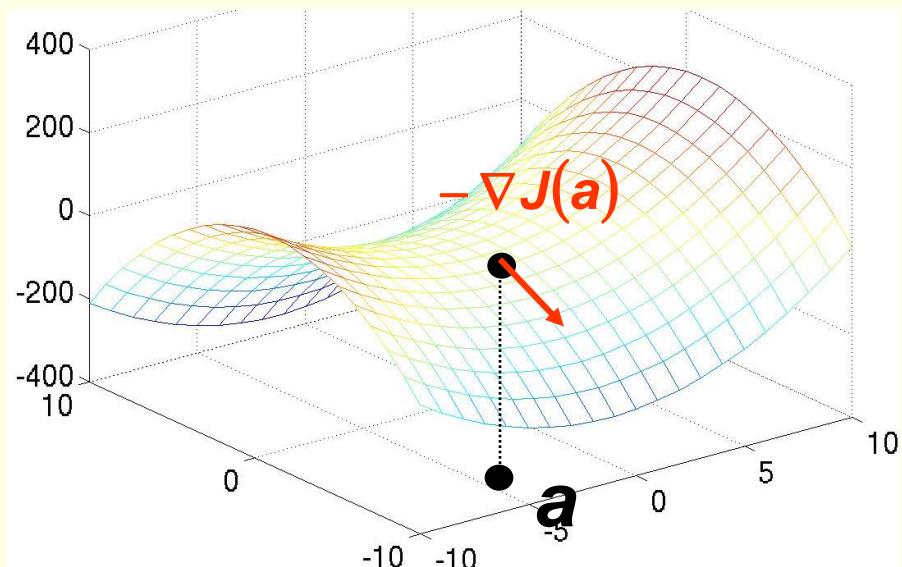
Optimization: Gradient Descent

- Gradient $\nabla J(x)$ points in direction of steepest increase of $J(x)$, and $-\nabla J(x)$ in direction of steepest decrease

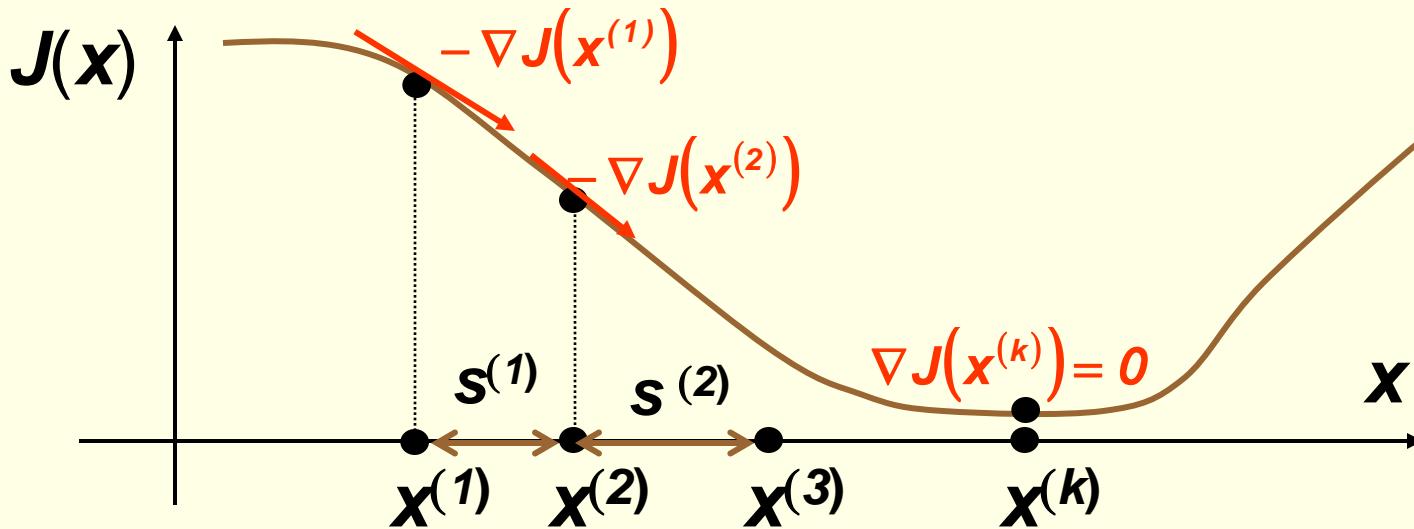
one dimension



two dimensions



Optimization: Gradient Descent



Gradient Descent for minimizing any function $J(\mathbf{x})$

set $k = 1$ **and** $\mathbf{x}^{(1)}$ to some initial guess for the weight vector

while $\eta^{(k)} |\nabla J(\mathbf{x}^{(k)})| > \varepsilon$

choose learning rate $\eta^{(k)}$

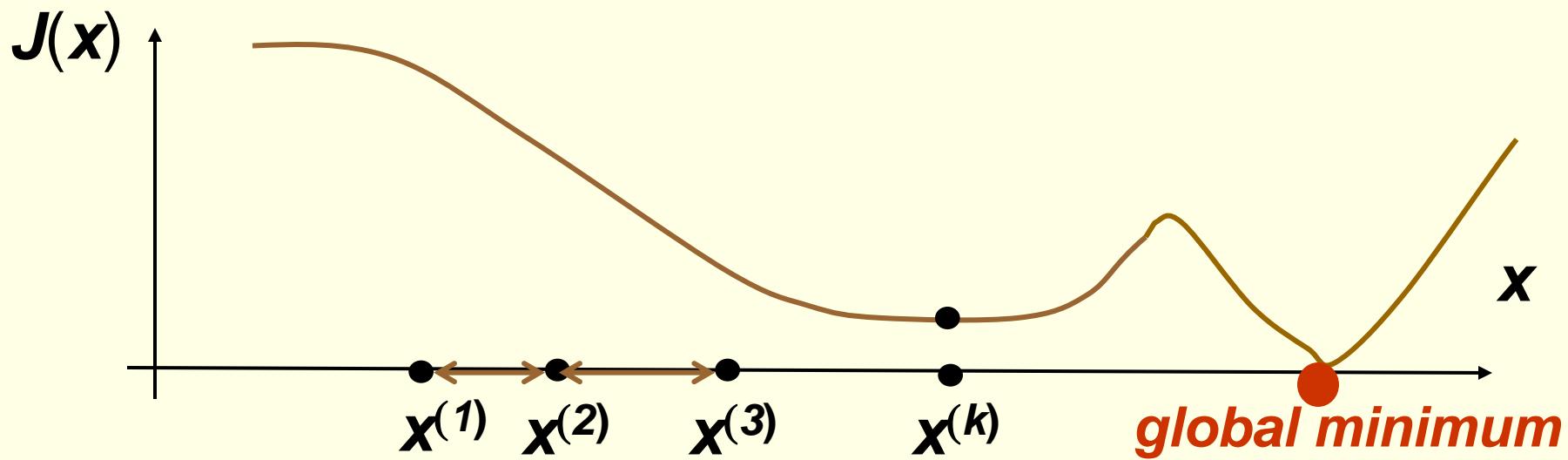
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta^{(k)} \nabla J(\mathbf{x})$$

(update rule)

$$k = k + 1$$

Optimization: Gradient Descent

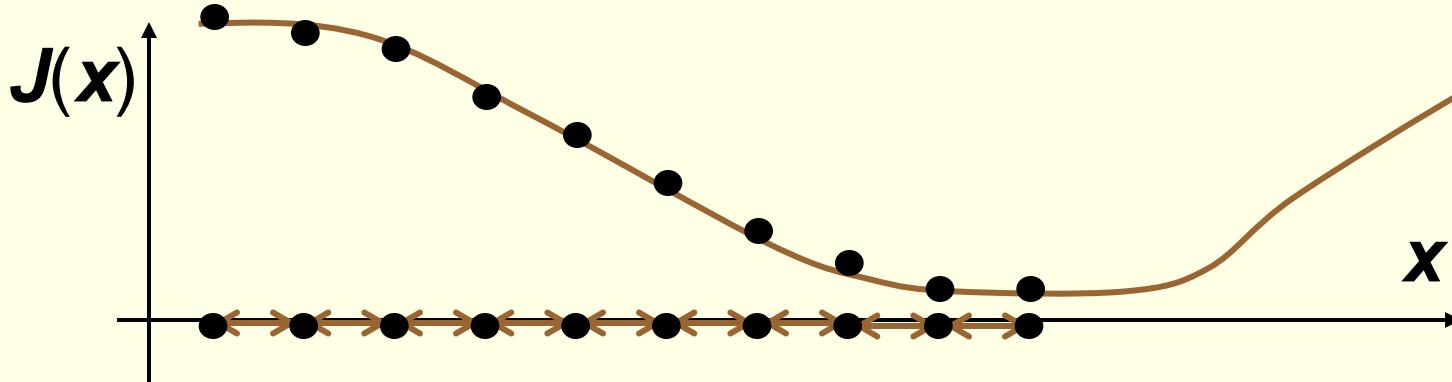
- Gradient descent is guaranteed to find only a local minimum



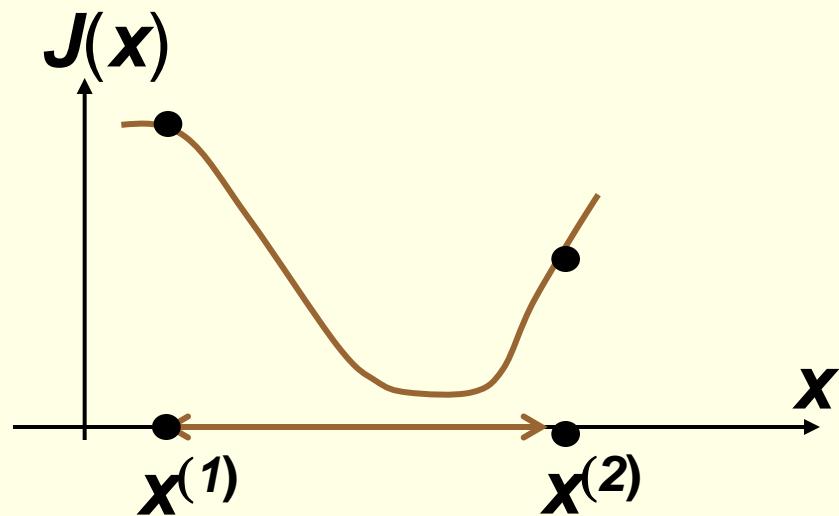
- Nevertheless gradient descent is very popular because it is simple and applicable to any function

Optimization: Gradient Descent

- Main issue: how to set parameter η (**learning rate**)
- If η is too small, need too many iterations



- If η is too large may overshoot the minimum and possibly never find it (if we keep overshooting)



LDF: Criterion Function

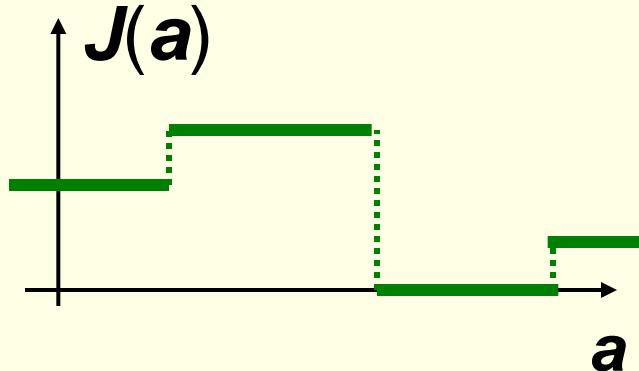
- Find weight vector \mathbf{a} s.t. for all samples $\mathbf{y}_1, \dots, \mathbf{y}_n$

$$\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^d \mathbf{a}_k y_i^{(k)} > 0$$

- Need criterion function $J(\mathbf{a})$ which is minimized when \mathbf{a} is a solution vector
- Let Y_M be the set of examples misclassified by \mathbf{a}
$$Y_M(\mathbf{a}) = \{ \text{sample } y_i \text{ s.t. } \mathbf{a}^t \mathbf{y}_i < 0 \}$$
- First natural choice: number of misclassified examples

$$J(\mathbf{a}) = |Y_M(\mathbf{a})|$$

- piecewise constant, gradient descent is useless



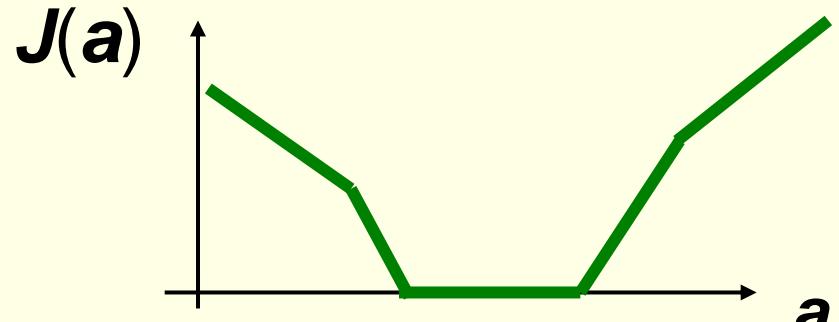
LDF: Perceptron Criterion Function

- Better choice: **Perceptron** criterion function

$$J_p(\mathbf{a}) = \sum_{y \in Y_M} (-\mathbf{a}^t \mathbf{y})$$

- If \mathbf{y} is misclassified, $\mathbf{a}^t \mathbf{y} \leq 0$
- Thus $J_p(\mathbf{a}) \geq 0$

- $J_p(\mathbf{a})$ is piecewise linear and thus suitable for gradient descent



LDF: Perceptron Batch Rule

$$J_p(\mathbf{a}) = \sum_{y \in Y_M} (-\mathbf{a}^t y)$$

- Gradient of $J_p(\mathbf{a})$ is $\nabla J_p(\mathbf{a}) = \sum_{y \in Y_M} (-y)$
 - Y_M are samples misclassified by $\mathbf{a}^{(k)}$
 - It is not possible to solve $\nabla J_p(\mathbf{a}) = \mathbf{0}$ analytically because of Y_M
- **Gradient decent batch update rule** for $J_p(\mathbf{a})$ is:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)} \sum_{y \in Y_M} y$$

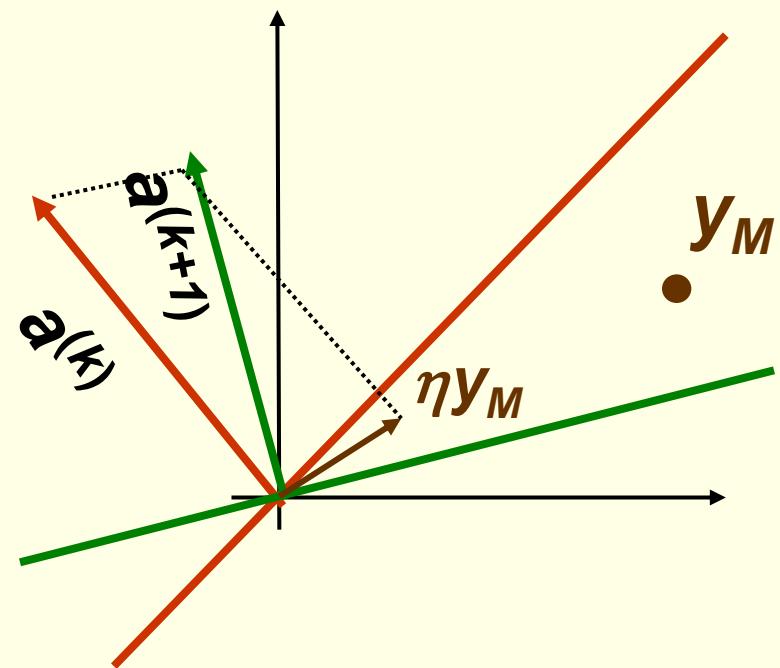
- It is called **batch** rule because it is based on all misclassified examples

LDF: Perceptron Single Sample Rule

- **Gradient decent single sample rule** for $J_p(\mathbf{a})$ is:

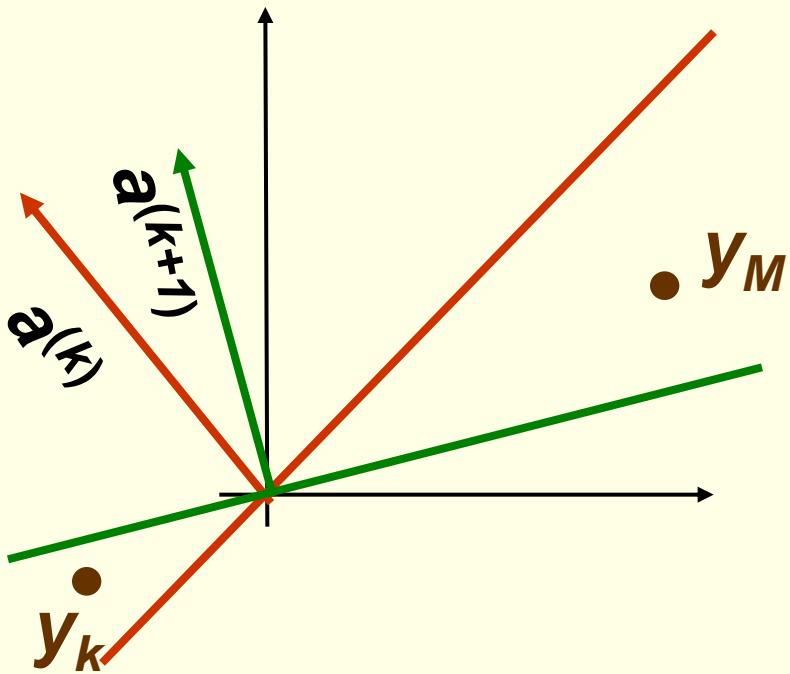
$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)} \mathbf{y}_M$$

- note that \mathbf{y}_M is one sample misclassified by $\mathbf{a}^{(k)}$
- must have a consistent way of visiting samples
- Geometric Interpretation:
 - \mathbf{y}_M misclassified by $\mathbf{a}^{(k)}$
 $(\mathbf{a}^{(k)})^t \mathbf{y}_M \leq 0$
 - \mathbf{y}_M is on the wrong side of decision hyperplane
 - adding $\eta \mathbf{y}_M$ to \mathbf{a} moves new decision hyperplane in the right direction with respect to \mathbf{y}_M

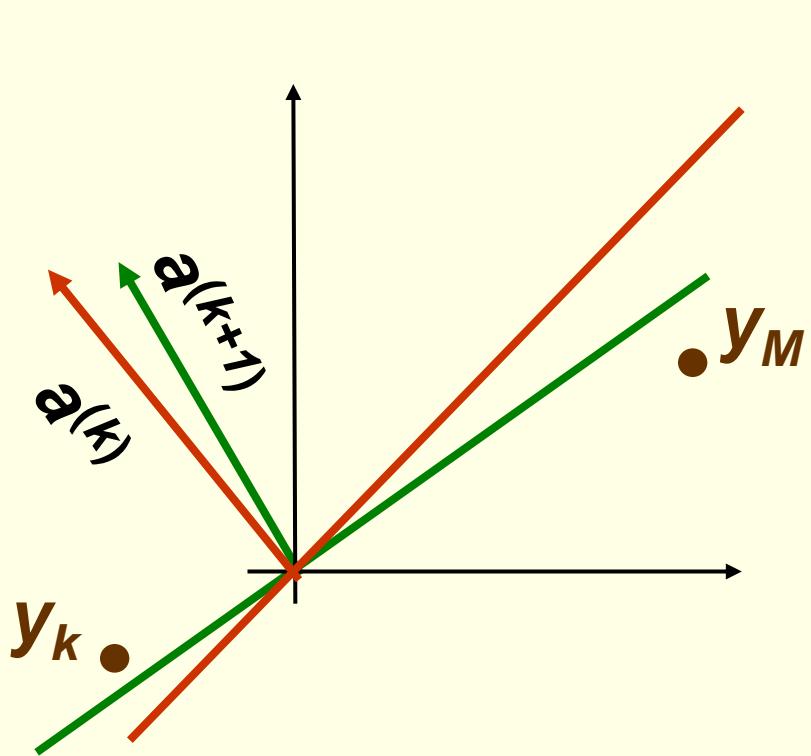


LDF: Perceptron Single Sample Rule

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)} y_M$$



η is too large, previously correctly classified sample y_k is now misclassified



η is too small, y_M is still misclassified

LDF: Perceptron Example

	features				grade
<i>name</i>	<i>good attendance?</i>	<i>tall?</i>	<i>sleeps in class?</i>	<i>chews gum?</i>	
Jane	yes (1)	yes (1)	no (-1)	no (-1)	A
Steve	yes (1)	yes (1)	yes (1)	yes (1)	F
Mary	no (-1)	no (-1)	no (-1)	yes (1)	F
Peter	yes (1)	no (-1)	no (-1)	yes (1)	A

- **class 1:** students who get grade A
- **class 2:** students who get grade F

LDF Example: Augment feature vector

	features						grade
<i>name</i>	extra	<i>good attendance?</i>	<i>tall?</i>	<i>sleeps in class?</i>	<i>chews gum?</i>		
Jane	1	yes (1)	yes (1)	no (-1)	no (-1)		A
Steve	1	yes (1)	yes (1)	yes (1)	yes (1)		F
Mary	1	no (-1)	no (-1)	no (-1)	yes (1)		F
Peter	1	yes (1)	no (-1)	no (-1)	yes (1)		A

- convert samples x_1, \dots, x_n to augmented samples y_1, \dots, y_n by adding a new dimension of value 1

LDF: Perform “Normalization”

	features					grade
<i>name</i>	extra	<i>good attendance?</i>	<i>tall?</i>	<i>sleeps in class?</i>	<i>chews gum?</i>	
Jane	1	yes (1)	yes (1)	no (-1)	no (-1)	A
Steve	-1	yes (-1)	yes (-1)	yes (-1)	yes (-1)	F
Mary	-1	no (1)	no (1)	no (1)	yes (-1)	F
Peter	1	yes (1)	no (-1)	no (-1)	yes (1)	A

- Replace all examples from class \mathbf{c}_2 by their negative

$$\mathbf{y}_i \rightarrow -\mathbf{y}_i \quad \forall \mathbf{y}_i \in \mathbf{c}_2$$

- Seek weight vector \mathbf{a} s.t. $\mathbf{a}^t \mathbf{y}_i > 0 \quad \forall \mathbf{y}_i$

LDF: Use Single Sample Rule

	features					grade
<i>name</i>	extra	good attendance?	tall?	sleeps in class?	chews gum?	
Jane	1	yes (1)	yes (1)	no (-1)	no (-1)	A
Steve	-1	yes (-1)	yes (-1)	yes (-1)	yes (-1)	F
Mary	-1	no (1)	no (1)	no (1)	yes (-1)	F
Peter	1	yes (1)	no (-1)	no (-1)	yes (1)	A

- Sample is misclassified if $\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^4 a_k y_i^{(k)} < 0$
- gradient descent single sample rule: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \eta^{(k)} \mathbf{y}_M$
- Set **fixed** learning rate to $\eta^{(k)} = 1$: $\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$

LDF: Gradient decent Example

- set equal initial weights $\mathbf{a}^{(1)} = [0.25, 0.25, 0.25, 0.25]$
- visit all samples sequentially, modifying the weights for after finding a misclassified example

<i>name</i>	$\mathbf{a}^t \mathbf{y}$	<i>misclassified?</i>
Jane	$0.25*1+0.25*1+0.25*1+0.25*(-1)+0.25*(-1) > 0$	no
Steve	$0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1)+0.25*(-1) < 0$	yes

- new weights

$$\begin{aligned}\mathbf{a}^{(2)} &= \mathbf{a}^{(1)} + \mathbf{y}_M = [0.25 \ 0.25 \ 0.25 \ 0.25 \ 0.25] + \\ &\quad + [-1 \ -1 \ -1 \ -1 \ -1] = \\ &= [-0.75 \ -0.75 \ -0.75 \ -0.75 \ -0.75]\end{aligned}$$

LDF: Gradient decent Example

$$\mathbf{a}^{(2)} = [-0.75 \ -0.75 \ -0.75 \ -0.75 \ -0.75]$$

<i>name</i>	<i>a^ty</i>	<i>misclassified?</i>
Mary	$-0.75*(-1)-0.75*1 \ -0.75 *1 \ -0.75 *1 \ -0.75*(-1) < 0$	yes

- new weights

$$\begin{aligned}\mathbf{a}^{(3)} &= \mathbf{a}^{(2)} + \mathbf{y}_M = [-0.75 \ -0.75 \ -0.75 \ -0.75 \ -0.75] + \\ &\quad + [-1 \ 1 \ 1 \ 1 \ -1] = \\ &= [-1.75 \ 0.25 \ 0.25 \ 0.25 \ -1.75]\end{aligned}$$

LDF: Gradient decent Example

$$\mathbf{a}^{(3)} = [-1.75 \ 0.25 \ 0.25 \ 0.25 \ -1.75]$$

<i>name</i>	<i>a^ty</i>	<i>misclassified?</i>
Peter	$-1.75 * 1 + 0.25 * 1 + 0.25 * (-1) + 0.25 * (-1) - 1.75 * 1 < 0$	yes

- new weights

$$\begin{aligned}\mathbf{a}^{(4)} &= \mathbf{a}^{(3)} + \mathbf{y}_M = [-1.75 \ 0.25 \ 0.25 \ 0.25 \ -1.75] + \\ &\quad + [1 \ 1 \ -1 \ -1 \ 1] = \\ &= [-0.75 \ 1.25 \ -0.75 \ -0.75 \ -0.75]\end{aligned}$$

LDF: Gradient decent Example

$$\mathbf{a}^{(4)} = [-0.75 \ 1.25 \ -0.75 \ -0.75 \ -0.75]$$

<i>name</i>	$\mathbf{a}^t \mathbf{y}$	<i>misclassified?</i>
Jane	$-0.75 * 1 + 1.25 * 1 - 0.75 * 1 - 0.75 * (-1) - 0.75 * (-1) > 0$	no
Steve	$-0.75 * (-1) + 1.25 * (-1) - 0.75 * (-1) - 0.75 * (-1) - 0.75 * (-1) > 0$	no
Mary	$-0.75 * (-1) + 1.25 * 1 - 0.75 * 1 - 0.75 * 1 - 0.75 * (-1) > 0$	no
Peter	$-0.75 * 1 + 1.25 * 1 - 0.75 * (-1) - 0.75 * (-1) - 0.75 * 1 > 0$	no

- Thus the discriminant function is

$$g(y) = -0.75 * y^{(0)} + 1.25 * y^{(1)} - 0.75 * y^{(2)} - 0.75 * y^{(3)} - 0.75 * y^{(4)}$$

- Converting back to the original features \mathbf{x} :

$$g(x) = 1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} - 0.75$$

LDF: Gradient decent Example

- Converting back to the original features x :

$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} > 0.75 \Rightarrow \text{grade A}$$

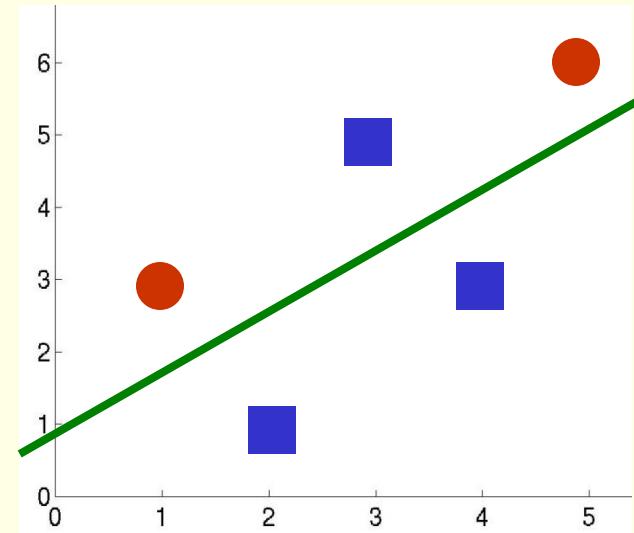
$$1.25 * x^{(1)} - 0.75 * x^{(2)} - 0.75 * x^{(3)} - 0.75 * x^{(4)} < 0.75 \Rightarrow \text{grade F}$$



- This is just one possible solution vector
- If we started with weights $a^{(1)} = [0, 0.5, 0.5, 0, 0]$, solution would be $[-1, 1.5, -0.5, -1, -1]$
 - $1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} > 1 \Rightarrow \text{grade A}$
 - $1.5 * x^{(1)} - 0.5 * x^{(2)} - x^{(3)} - x^{(4)} < 1 \Rightarrow \text{grade F}$
- In this solution, being tall is the least important feature

LDF: Nonseparable Example

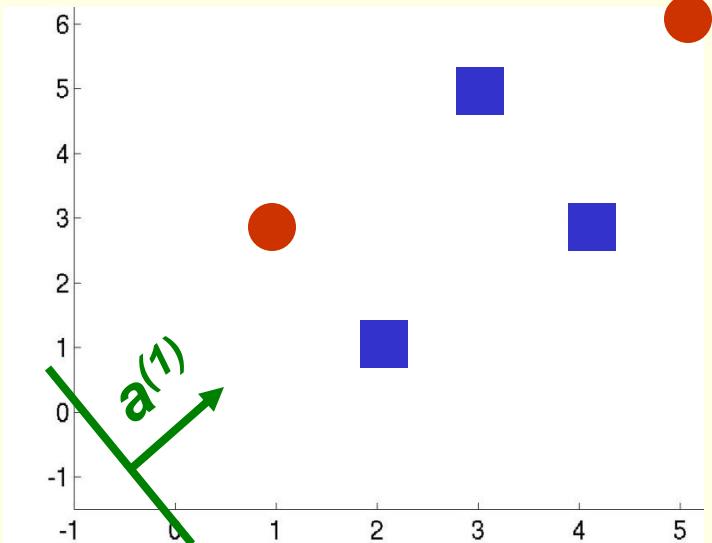
- Suppose we have 2 features and samples are:
 - Class 1: [2,1], [4,3], [3,5]
 - Class 2: [1,3] and [5,6]
- These samples are not separable by a line
- Still would like to get approximate separation by a line, good choice is shown in green
 - some samples may be “noisy”, and it’s ok if they are on the wrong side of the line
- Get y_1, y_2, y_3, y_4 by adding extra feature and “normalizing”



$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

LDF: Nonseparable Example

- Let's apply Perceptron single sample algorithm
- initial equal weights $\mathbf{a}^{(1)} = [1 \ 1 \ 1]$
 - this is line $x^{(1)} + x^{(2)} + 1 = 0$
- fixed learning rate $\eta = 1$
$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + y_M$$



$$y_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad y_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad y_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $y_1^T \mathbf{a}^{(1)} = [1 \ 1 \ 1] * [1 \ 2 \ 1]^T > 0 \quad \checkmark$
- $y_2^T \mathbf{a}^{(1)} = [1 \ 1 \ 1] * [1 \ 4 \ 3]^T > 0 \quad \checkmark \square$
- $y_3^T \mathbf{a}^{(1)} = [1 \ 1 \ 1] * [1 \ 3 \ 5]^T > 0 \quad \checkmark \square$

LDF: Nonseparable Example

$$\mathbf{a}^{(1)} = [1 \ 1 \ 1] \quad \mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$$

$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{y}_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{y}_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{y}_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{y}_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

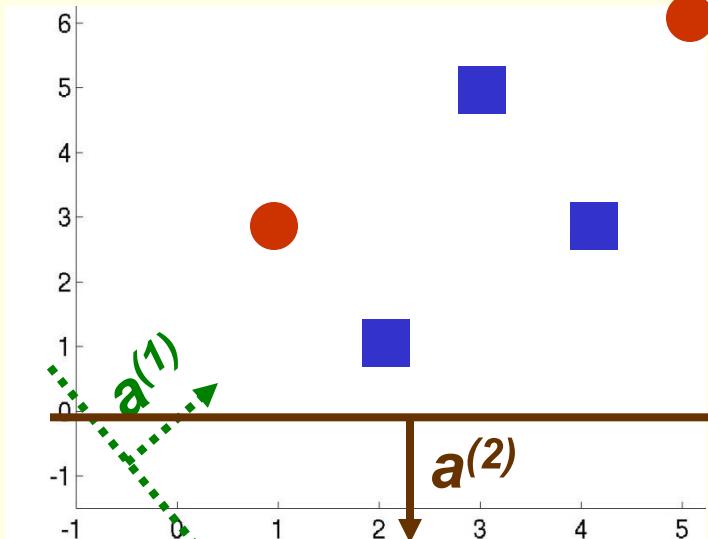
- $\mathbf{y}_4^T \mathbf{a}^{(1)} = [1 \ 1 \ 1]^* [-1 \ -1 \ -3]^t = -5 < 0$

$$\mathbf{a}^{(2)} = \mathbf{a}^{(1)} + \mathbf{y}_M = [1 \ 1 \ 1] + [-1 \ -1 \ -3] = [0 \ 0 \ -2]$$

- $\mathbf{y}_5^T \mathbf{a}^{(2)} = [0 \ 0 \ -2]^* [-1 \ -5 \ -6]^t = 12 > 0 \quad \checkmark \square$

- $\mathbf{y}_1^T \mathbf{a}^{(2)} = [0 \ 0 \ -2]^* [1 \ 2 \ 1]^t < 0 \square$

$$\mathbf{a}^{(3)} = \mathbf{a}^{(2)} + \mathbf{y}_M = [0 \ 0 \ -2] + [1 \ 2 \ 1] = [1 \ 2 \ -1]$$



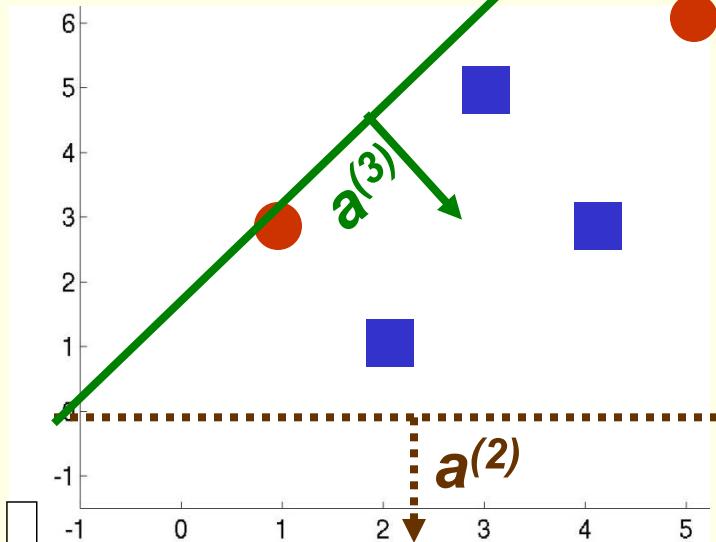
LDF: Nonseparable Example

$$\mathbf{a}^{(3)} = [1 \ 2 \ -1] \quad \mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$$

$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{y}_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{y}_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{y}_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{y}_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $\mathbf{y}_2^T \mathbf{a}^{(3)} = [1 \ 4 \ 3]^* [1 \ 2 \ -1]^t = 6 > 0 \quad \checkmark \square$
- $\mathbf{y}_3^T \mathbf{a}^{(3)} = [1 \ 3 \ 5]^* [1 \ 2 \ -1]^t > 0 \quad \checkmark \square$
- $\mathbf{y}_4^T \mathbf{a}^{(3)} = [-1 \ -1 \ -3]^* [1 \ 2 \ -1]^t = 0 \square$

$$\mathbf{a}^{(4)} = \mathbf{a}^{(3)} + \mathbf{y}_M = [1 \ 2 \ -1] + [-1 \ -1 \ -3] = [0 \ 1 \ -4]$$



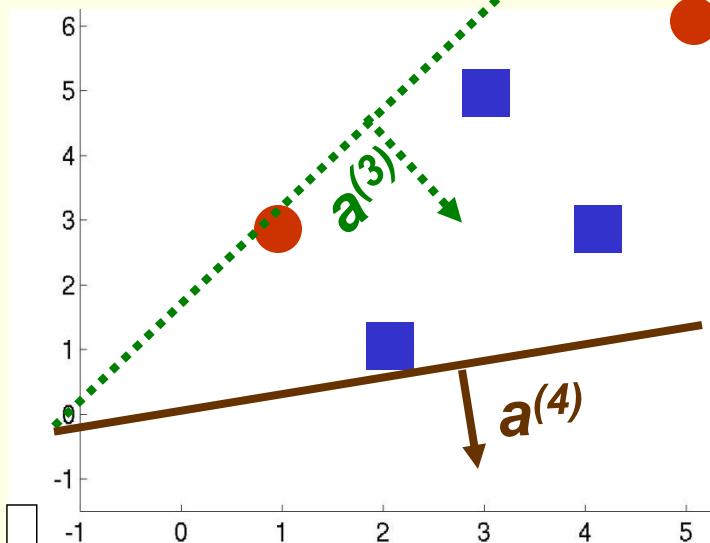
LDF: Nonseparable Example

$$\mathbf{a}^{(4)} = [0 \ 1 \ -4] \quad \mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} + \mathbf{y}_M$$

$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \quad \mathbf{y}_2 = \begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \quad \mathbf{y}_3 = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} \quad \mathbf{y}_4 = \begin{bmatrix} -1 \\ -1 \\ -3 \end{bmatrix} \quad \mathbf{y}_5 = \begin{bmatrix} -1 \\ -5 \\ -6 \end{bmatrix}$$

- $\mathbf{y}_2^T \mathbf{a}^{(3)} = [1 \ 4 \ 3]^* [1 \ 2 \ -1]^t = 6 > 0 \quad \checkmark \quad \square$
- $\mathbf{y}_3^T \mathbf{a}^{(3)} = [1 \ 3 \ 5]^* [1 \ 2 \ -1]^t > 0 \quad \checkmark \quad \square$
- $\mathbf{y}_4^T \mathbf{a}^{(3)} = [-1 \ -1 \ -3]^* [1 \ 2 \ -1]^t = 0 \quad \square$

$$\mathbf{a}^{(4)} = \mathbf{a}^{(3)} + \mathbf{y}_M = [1 \ 2 \ -1] + [-1 \ -1 \ -3] = [0 \ 1 \ -4]$$



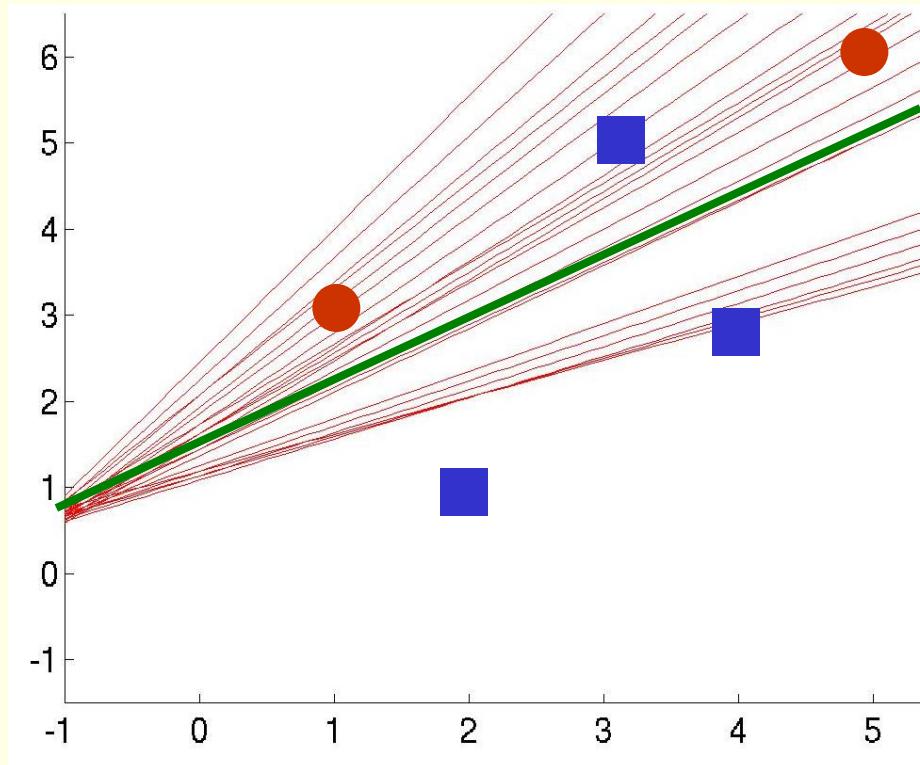
LDF: Nonseparable Example

- we can continue this forever
 - there is no solution vector \mathbf{a} satisfying for all $i \square$

$$\mathbf{a}^t \mathbf{y}_i = \sum_{k=0}^5 \mathbf{a}_k y_i^{(k)} > 0$$

- need to stop but at a good point:

- solutions at iterations 900 through 915.
Some are good
some are not.
- How do we stop at a
good solution? \square



LDF: Convergence of Perceptron rules

- If classes are linearly separable, and use fixed learning rate, that is for some constant c , $\eta^{(k)}=c$
 - *both single sample and batch perceptron rules converge to a correct solution* (could be any a in the solution space)
- If classes are not linearly separable:
 - algorithm does not stop, it keeps looking for solution which does not exist
 - by choosing appropriate learning rate, can always ensure convergence: $\eta^{(k)} \rightarrow 0$ as $k \rightarrow \infty$
 - for example inverse linear learning rate: $\eta^{(k)} = \frac{\eta^{(1)}}{k}$
 - for inverse linear learning rate convergence in the linearly separable case can also be proven
 - no guarantee that we stopped at a good point, but is popular in practice.

LDF: Perceptron Rule and Gradient decent

- Linearly separable data
 - perceptron rule with gradient decent works well
- Linearly non-separable data
 - need to stop perceptron rule algorithm at a good point, this maybe tricky

Batch Rule

- Smoother gradient because all samples are used

Single Sample Rule

- easier to analyze
- Concentrates more than necessary on any isolated “noisy” training examples

Minimum Squared Error

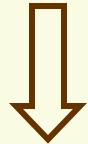
LDF: Minimum Squared-Error Procedures

- Idea: convert to easier and better understood problem

Perceptron

$\mathbf{a}^t \mathbf{y}_i > 0$ for all samples \mathbf{y}_i

solve system of linear inequalities

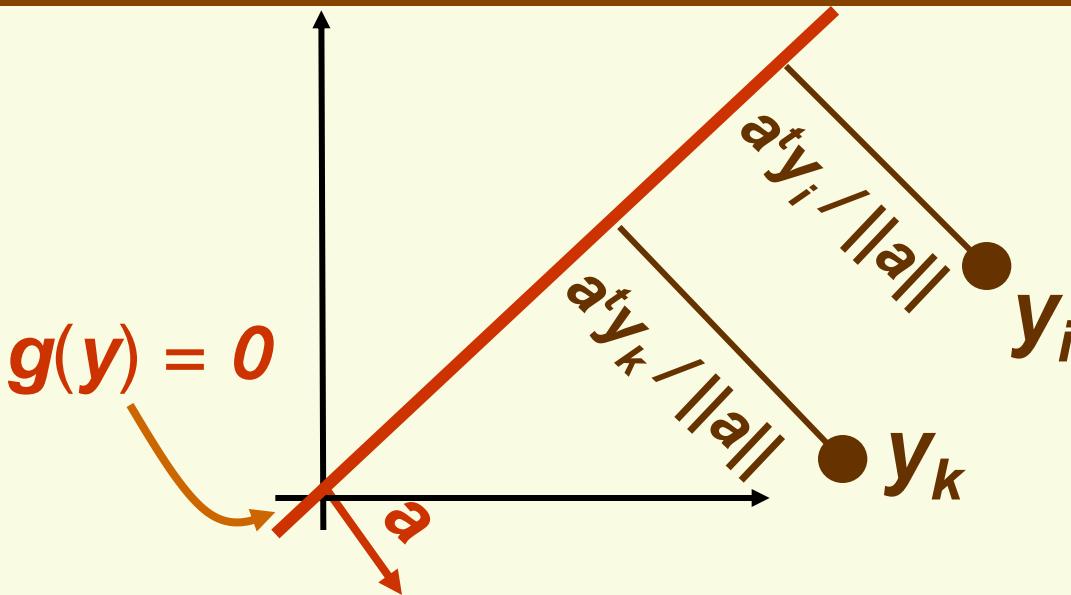


$\mathbf{a}^t \mathbf{y}_i = b_i$ for all samples \mathbf{y}_i

solve system of linear equations

- MSE procedure
 - Choose **positive** constants b_1, b_2, \dots, b_n
 - try to find weight vector \mathbf{a} s.t. $\mathbf{a}^t \mathbf{y}_i = b_i$ for all samples \mathbf{y}_i
 - If we can find weight vector \mathbf{a} such that $\mathbf{a}^t \mathbf{y}_i = b_i$ for all samples \mathbf{y}_i , then \mathbf{a} is a solution because b_i 's are positive
 - consider all the samples (not just the misclassified ones)

LDF: MSE Margins



- Since we want $\mathbf{a}^t \mathbf{y}_i = b_i$, we expect sample \mathbf{y}_i to be at distance b_i from the separating hyperplane (normalized by $\|\mathbf{a}\|$)
- Thus b_1, b_2, \dots, b_n give relative expected distances or “margins” of samples from the hyperplane
- Should make b_i small if sample i is expected to be near separating hyperplane, and make b_i larger otherwise
- In the absence of any additional information, there are good reasons to set $b_1 = b_2 = \dots = b_n = 1$

LDF: MSE Matrix Notation

- Need to solve n equations

$$\begin{cases} \mathbf{a}^t \mathbf{y}_1 = \mathbf{b}_1 \\ \vdots \\ \mathbf{a}^t \mathbf{y}_n = \mathbf{b}_n \end{cases}$$

- Introduce matrix notation:

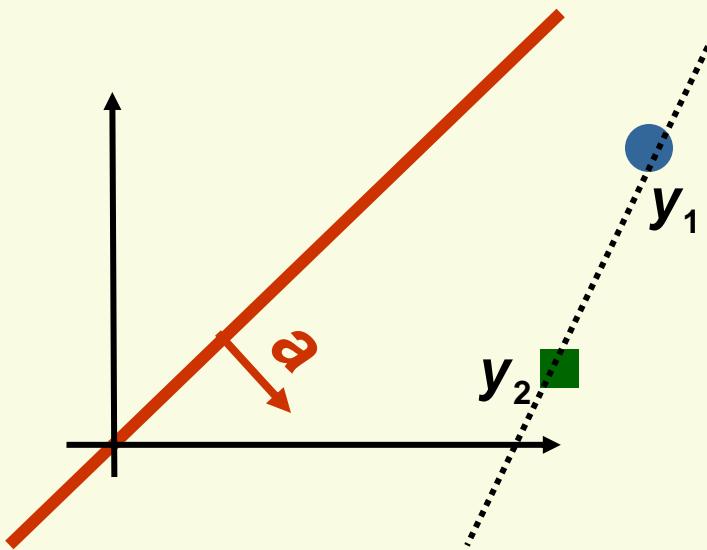
$$\left[\begin{array}{cccc} \mathbf{y}_1^{(0)} & \mathbf{y}_1^{(1)} & \cdots & \mathbf{y}_1^{(d)} \\ \mathbf{y}_2^{(0)} & \mathbf{y}_2^{(1)} & \cdots & \mathbf{y}_2^{(d)} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \mathbf{y}_n^{(0)} & \mathbf{y}_n^{(1)} & \cdots & \mathbf{y}_n^{(d)} \end{array} \right] \left[\begin{array}{c} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_d \end{array} \right] = \left[\begin{array}{c} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \vdots \\ \mathbf{b}_n \end{array} \right]$$

$\underbrace{\hspace{10em}}_{\mathbf{Y}}$ $\underbrace{\hspace{1.5em}}_{\mathbf{a}}$ $\underbrace{\hspace{1.5em}}_{\mathbf{b}}$

- Thus need to solve a linear system $\mathbf{Ya} = \mathbf{b}$

LDF: Exact Solution is Rare

- Thus need to solve a linear system $\mathbf{Y}\mathbf{a} = \mathbf{b}$
 - \mathbf{Y} is an n by $(d+1)$ matrix
- Exact solution can be found only if \mathbf{Y} is nonsingular and square, in which case the inverse \mathbf{Y}^{-1} exists
 - $\mathbf{a} = \mathbf{Y}^{-1}\mathbf{b}$
 - (number of samples) = (number of features + 1)
 - almost never happens in practice
 - in this case, guaranteed to find the separating hyperplane



LDF: Approximate Solution

- Typically \mathbf{Y} is overdetermined, that is it has more rows (examples) than columns (features)
 - If it has more features than examples, should reduce dimensionality

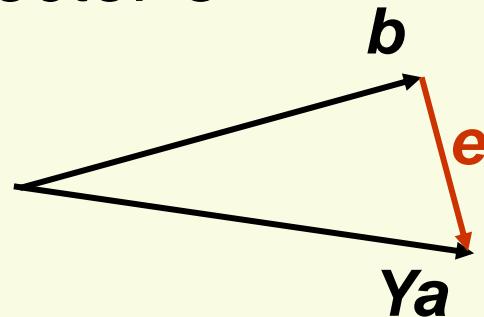
$$\begin{array}{|c|c|c|} \hline \mathbf{Y} & \mathbf{a} & = \mathbf{b} \\ \hline \end{array}$$

- Need $\mathbf{Ya} = \mathbf{b}$, but no exact solution exists for an overdetermined system of equation
 - More equations than unknowns
- Find an approximate solution \mathbf{a} , that is $\mathbf{Ya} \approx \mathbf{b}$
 - Note that approximate solution \mathbf{a} **does not** necessarily give the separating hyperplane in the separable case
 - But hyperplane corresponding to \mathbf{a} may still be a good solution, especially if there is no separating hyperplane

LDF: MSE Criterion Function

- Minimum squared error approach: find \mathbf{a} which minimizes the length of the error vector \mathbf{e}

$$\mathbf{e} = \mathbf{Y}\mathbf{a} - \mathbf{b}$$



- Thus minimize the *minimum squared error* criterion function:

$$J_s(\mathbf{a}) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2 = \sum_{i=1}^n (\mathbf{a}^t \mathbf{y}_i - b_i)^2$$

- Unlike the perceptron criterion function, we can optimize the minimum squared error criterion function analytically by setting the gradient to 0

LDF: Optimizing $J_s(a)$

$$J_s(a) = \|\mathbf{Y}a - \mathbf{b}\|^2 = \sum_{i=1}^n (a^t y_i - b_i)^2$$

- Let's compute the gradient:

$$\nabla J_s(a) = \begin{bmatrix} \frac{\partial J_s}{\partial a_0} \\ \vdots \\ \frac{\partial J_s}{\partial a_d} \end{bmatrix} = 2\mathbf{Y}^t(\mathbf{Y}a - \mathbf{b})$$

- Setting the gradient to 0:

$$2\mathbf{Y}^t(\mathbf{Y}a - \mathbf{b}) = \mathbf{0} \Rightarrow \mathbf{Y}^t\mathbf{Y}a = \mathbf{Y}^t\mathbf{b}$$

LDF: Pseudo Inverse Solution

- Matrix $\mathbf{Y}^t \mathbf{Y}$ is square (it has $d+1$ rows and columns) and it is often non-singular
- If $\mathbf{Y}^t \mathbf{Y}$ is non-singular, its inverse exists and we can solve for \mathbf{a} uniquely:

$$\mathbf{a} = \boxed{(\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t} \mathbf{b}$$

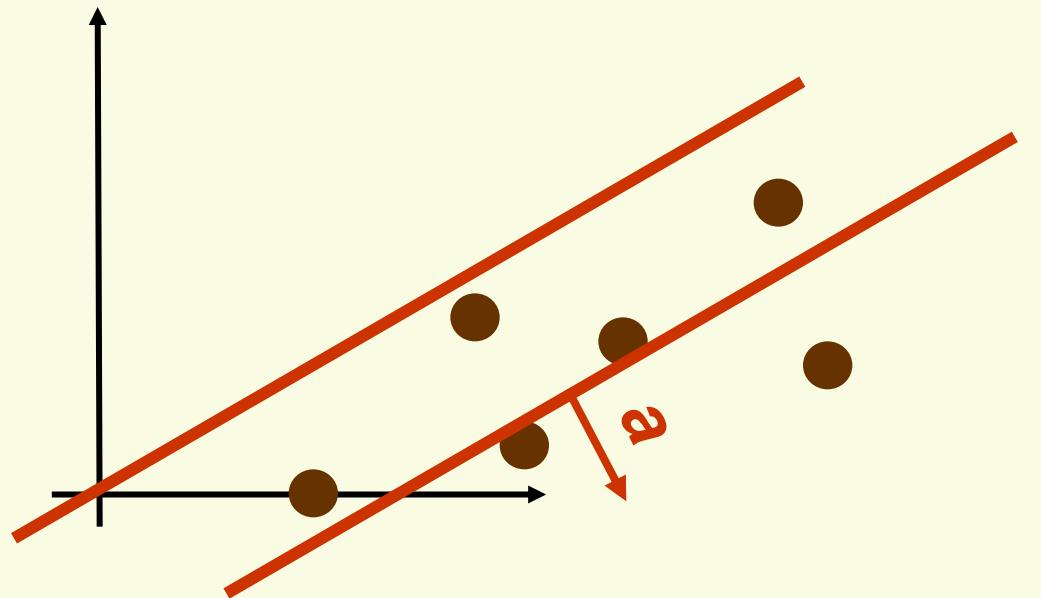
pseudo inverse of Y

$$((\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t) \mathbf{Y} = (\mathbf{Y}^t \mathbf{Y})^{-1} (\mathbf{Y}^t \mathbf{Y}) = I$$

LDF: Minimum Squared-Error Procedures

- If $b_1 = \dots = b_n = 1$, MSE procedure is equivalent to finding a hyperplane of best fit through the samples y_1, \dots, y_n

$$J_s(\mathbf{a}) = \|\mathbf{Y}\mathbf{a} - \mathbf{1}_n\|^2$$
$$\mathbf{1}_n = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \quad n$$



- Then we shift this line to the origin, if this line was a good fit, all samples will be classified correctly

LDF: Minimum Squared-Error Procedures

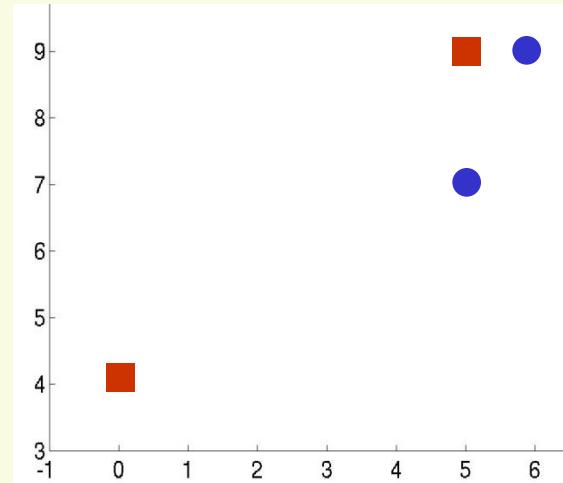
- Only guaranteed the separating hyperplane if $\mathbf{Y}\mathbf{a} > 0$
 - that is if all elements of vector $\mathbf{Y}\mathbf{a} = \begin{bmatrix} \mathbf{a}^t \mathbf{y}_1 \\ \vdots \\ \mathbf{a}^t \mathbf{y}_n \end{bmatrix}$ are positive
- We have $\mathbf{Y}\mathbf{a} \approx \mathbf{b}$
- That is $\mathbf{Y}\mathbf{a} = \begin{bmatrix} \mathbf{b}_1 + \varepsilon_1 \\ \vdots \\ \mathbf{b}_n + \varepsilon_n \end{bmatrix}$ where ε may be negative
 - If $\varepsilon_1, \dots, \varepsilon_n$ are small relative to $\mathbf{b}_1, \dots, \mathbf{b}_n$, then each element of $\mathbf{Y}\mathbf{a}$ is positive, and \mathbf{a} gives a separating hyperplane
 - If approximation is not good, ε_i may be large and negative, for some i , thus $\mathbf{b}_i + \varepsilon_i$ will be negative and \mathbf{a} is not a separating hyperplane
- Thus in linearly separable case, least squares solution \mathbf{a} does *not necessarily* give a separating hyperplane
- But it will give a “reasonable” hyperplane

LDF: Minimum Squared-Error Procedures

- We are free to choose \mathbf{b} . May be tempted to make \mathbf{b} large as a way to insure $\mathbf{Y}\mathbf{a} \approx \mathbf{b} > 0$
- Does not work
 - Let β be a scalar, let's try $\beta\mathbf{b}$ instead of \mathbf{b}
 - if \mathbf{a}^* is a least squares solution to $\mathbf{Y}\mathbf{a} = \mathbf{b}$, then for any scalar β , least squares solution to $\mathbf{Y}\mathbf{a} = \beta\mathbf{b}$ is $\beta\mathbf{a}^*$
$$\begin{aligned} \underset{\mathbf{a}}{\operatorname{argmin}} \|\mathbf{Y}\mathbf{a} - \beta\mathbf{b}\|^2 &= \underset{\mathbf{a}}{\operatorname{argmin}} \beta^2 \|\mathbf{Y}(\mathbf{a}/\beta) - \mathbf{b}\|^2 \\ &= \underset{\mathbf{a}}{\operatorname{argmin}} \|\mathbf{Y}(\mathbf{a}/\beta) - \mathbf{b}\|^2 = \beta\mathbf{a}^* \end{aligned}$$
 - thus if for some i th element of $\mathbf{Y}\mathbf{a}$ is less than 0, that is $y_{it}^t \mathbf{a} < 0$, then $y_{it}^t (\beta\mathbf{a}) < 0$,
- Relative difference between components of \mathbf{b} matters, but not the size of each individual component

LDF: Example

- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 4)
- Set vectors y_1, y_2, y_3, y_4 by adding extra feature and “normalizing”



$$y_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad y_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad y_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad y_4 = \begin{bmatrix} -1 \\ 0 \\ -4 \end{bmatrix}$$

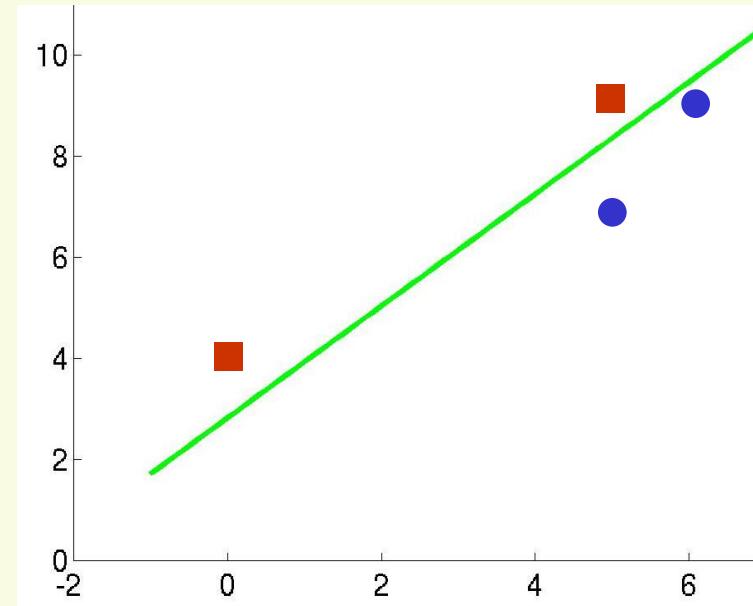
- Matrix Y is then

$$Y = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -4 \end{bmatrix}$$

LDF: Example

- Choose $\mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
- In matlab, $\mathbf{a} = \mathbf{Y}\backslash\mathbf{b}$ solves the least squares problem

$$\mathbf{a} = \begin{bmatrix} 2.7 \\ 1.0 \\ -0.9 \end{bmatrix}$$



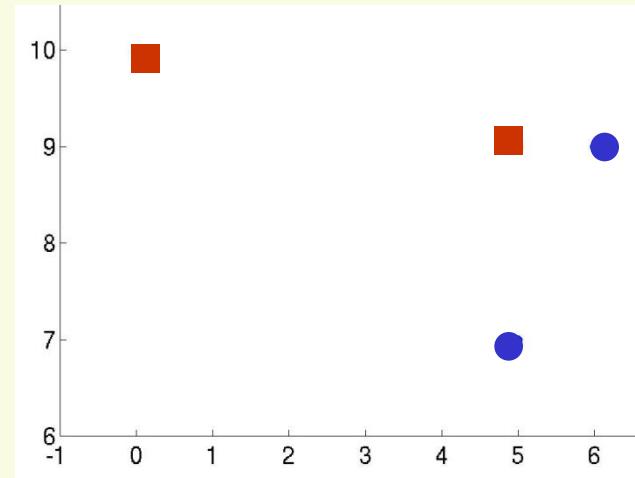
- Note \mathbf{a} is an approximation to $\mathbf{Ya} = \mathbf{b}$, since no exact solution exists

$$\mathbf{Ya} = \begin{bmatrix} 0.4 \\ 1.3 \\ 0.6 \\ 1.1 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

- This solution does give a separating hyperplane since $\mathbf{Ya} > 0$

LDF: Example

- Class 1: (6 9), (5 7)
- Class 2: (5 9), (0 10)
- The last sample is very far compared to others from the separating hyperplane



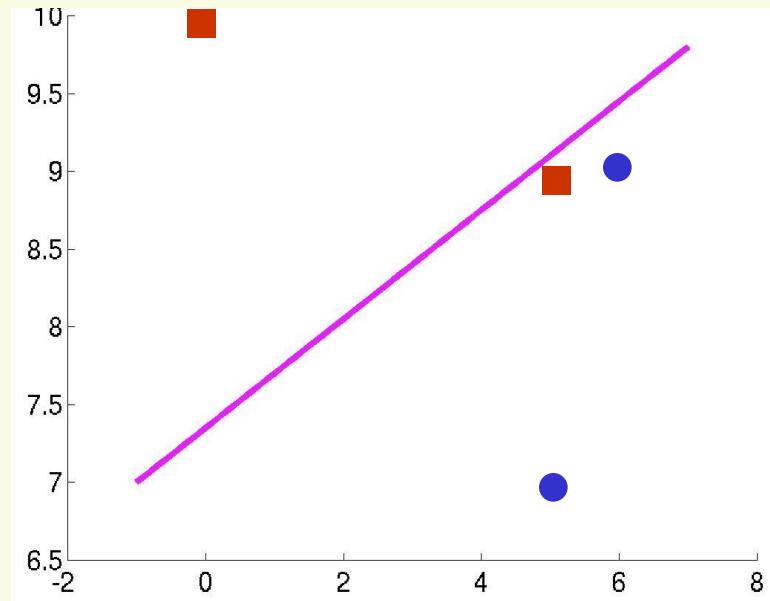
$$\mathbf{y}_1 = \begin{bmatrix} 1 \\ 6 \\ 9 \end{bmatrix} \quad \mathbf{y}_2 = \begin{bmatrix} 1 \\ 5 \\ 7 \end{bmatrix} \quad \mathbf{y}_3 = \begin{bmatrix} -1 \\ -5 \\ -9 \end{bmatrix} \quad \mathbf{y}_4 = \begin{bmatrix} -1 \\ 0 \\ -10 \end{bmatrix}$$

- Matrix $\mathbf{Y} = \begin{bmatrix} 1 & 6 & 9 \\ 1 & 5 & 7 \\ -1 & -5 & -9 \\ -1 & 0 & -10 \end{bmatrix}$

LDF: Example

- Choose $\mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$
- In matlab, $\mathbf{a} = \mathbf{Y}\backslash\mathbf{b}$ solves the least squares problem

$$\mathbf{a} = \begin{bmatrix} 3.2 \\ 0.2 \\ -0.4 \end{bmatrix}$$



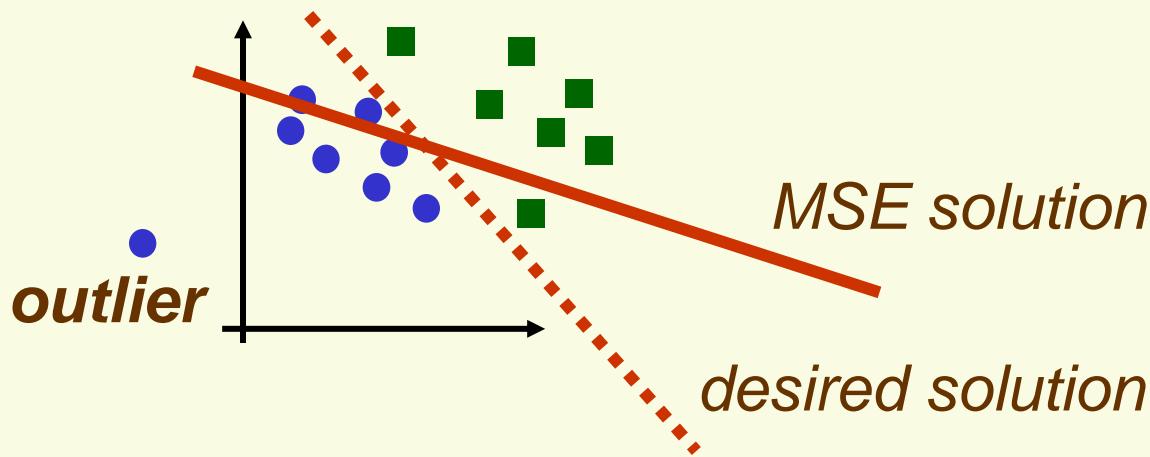
- Note \mathbf{a} is an approximation to $\mathbf{Y}\mathbf{a} = \mathbf{b}$, since no exact solution exists

$$\mathbf{Y}\mathbf{a} = \begin{bmatrix} 0.2 \\ 0.9 \\ -0.04 \\ 1.16 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

- This solution does not give a separating hyperplane since $\mathbf{a}^t \mathbf{y}_3 < 0$

LDF: Example

- MSE pays too much attention to isolated “noisy” examples (such examples are called outliers)



- No problems with convergence though, and solution it gives ranges from reasonable to good

LDF: Example

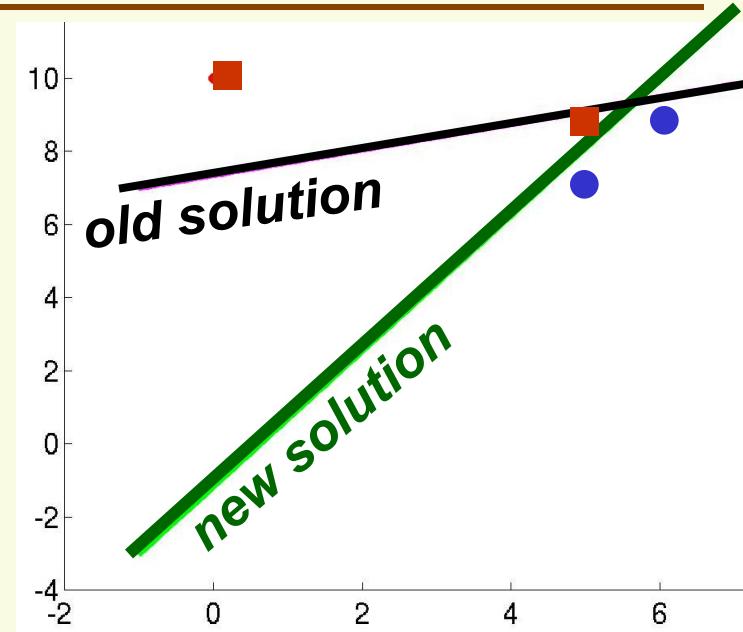
- we know that 4th point is far from separating hyperplane
 - In practice we don't know this

$$\text{Thus appropriate } \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$$

- In Matlab, solve $\mathbf{a} = \mathbf{Y} \setminus \mathbf{b}$

$$\mathbf{a} = \begin{bmatrix} -1.1 \\ 1.7 \\ -0.9 \end{bmatrix}$$

- Note \mathbf{a} is an approximation to $\mathbf{Y}\mathbf{a} = \mathbf{b}$, $\mathbf{Y}\mathbf{a} = \begin{bmatrix} 0.9 \\ 1.0 \\ 0.8 \\ 10.0 \end{bmatrix} \neq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 10 \end{bmatrix}$
- This solution does give the separating hyperplane since $\mathbf{Y}\mathbf{a} > 0$



LDF: Gradient Descent for MSE solution

$$J_s(\mathbf{a}) = \|\mathbf{Y}\mathbf{a} - \mathbf{b}\|^2$$

- May wish to find MSE solution by gradient descent:
 1. Computing the inverse of $\mathbf{Y}^t\mathbf{Y}$ may be too costly
 2. $\mathbf{Y}^t\mathbf{Y}$ may be close to singular if samples are highly correlated (rows of \mathbf{Y} are almost linear combinations of each other)
 - computing the inverse of $\mathbf{Y}^t\mathbf{Y}$ is not numerically stable
- In the beginning of the lecture, computed the gradient:

$$\nabla J_s(\mathbf{a}) = 2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b})$$

LDF: Widrow-Hoff Procedure

$$\nabla J_s(\mathbf{a}) = 2\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b})$$

- Thus the update rule for gradient descent:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \eta^{(k)} \mathbf{Y}^t(\mathbf{Y}\mathbf{a}^{(k)} - \mathbf{b})$$

- If $\eta^{(k)} = \eta^{(1)} / k$ weight vector $\mathbf{a}^{(k)}$ converges to the MSE solution \mathbf{a} , that is $\mathbf{Y}^t(\mathbf{Y}\mathbf{a} - \mathbf{b}) = 0$
- *Widrow-Hoff procedure* reduces storage requirements by considering single samples sequentially:

$$\mathbf{a}^{(k+1)} = \mathbf{a}^{(k)} - \eta^{(k)} \mathbf{y}_i (\mathbf{y}_i^t \mathbf{a}^{(k)} - b_i)$$

LDF: MSE for Multiple Classes

- Suppose we have m classes
- Define m linear discriminant functions

$$g_i(\mathbf{x}) = \mathbf{w}_i^t \mathbf{x} + w_{i0} \quad i = 1, \dots, m$$

- Given \mathbf{x} , assign class c_i if

$$g_i(\mathbf{x}) \geq g_j(\mathbf{x}) \quad \forall j \neq i$$

- Such classifier is called a *linear machine*
- A linear machine divides the feature space into c decision regions, with $g_i(\mathbf{x})$ being the largest discriminant if \mathbf{x} is in the region R_i

LDF: MSE for Multiple Classes

- For each class i , find weight vector \mathbf{a}_i , s.t.

$$\begin{cases} \mathbf{a}_i^t \mathbf{y} = 1 & \forall \mathbf{y} \in \text{class } i \\ \mathbf{a}_i^t \mathbf{y} = 0 & \forall \mathbf{y} \notin \text{class } i \end{cases}$$

- Let \mathbf{Y}_i be matrix whose rows are samples from class i , so it has $d+1$ columns and n_i rows
- Let's pile all samples in n by $d+1$ matrix \mathbf{Y} :

$$\mathbf{Y} = \begin{bmatrix} \mathbf{Y}_1 \\ \mathbf{Y}_2 \\ \vdots \\ \mathbf{Y}_m \end{bmatrix} = \begin{bmatrix} \text{sample from class1} \\ \text{sample from class1} \\ \vdots \\ \text{sample from class}m \\ \text{sample from class}m \end{bmatrix}$$

LDF: MSE for Multiple Classes

- Let \mathbf{b}_i be a column vector of length n which is 0 everywhere except rows corresponding to samples from class i , where it is 1 :

$$\mathbf{b}_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

*rows corresponding
to samples from class i*

LDF: MSE for Multiple Classes

- Let's pile all \mathbf{b}_i as columns in n by c matrix B

$$B = [\mathbf{b}_1 \ \dots \ \mathbf{b}_n]$$

- Let's pile all \mathbf{a}_i as columns in $d + 1$ by m matrix A

$$A = [\mathbf{a}_1 \ \dots \ \mathbf{a}_m] = \begin{bmatrix} \text{weights } \mathbf{a}_1 \\ \text{weights } \mathbf{a}_2 \\ \vdots \\ \text{weights } \mathbf{a}_m \end{bmatrix}$$

- m LSE problems can be represented in $YA = B$:

$$\begin{bmatrix} \text{sample from class1} \\ \text{sample from class1} \\ \text{sample from class2} \\ \text{sample from class3} \\ \text{sample from class3} \\ \text{sample from class3} \end{bmatrix} \begin{bmatrix} \text{weights for c1} \\ \text{weights for c2} \\ \text{weights for c3} \end{bmatrix} \parallel \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Y **A** **B**

LDF: MSE for Multiple Classes

- Our objective function is:

$$J(\mathbf{A}) = \sum_{i=1}^m \|\mathbf{Y}\mathbf{a}_i - \mathbf{b}_i\|^2$$

- $J(\mathbf{A})$ is minimized with the use of pseudoinverse

$$\mathbf{A} = (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y} \mathbf{B}$$

LDF: Summary

- ***Perceptron*** procedures

- find a separating hyperplane in the linearly separable case,
- do not converge in the non-separable case
- can force convergence by using a decreasing learning rate, but are not guaranteed a reasonable stopping point

- ***MSE*** procedures

- converge in separable and not separable case
- may not find separating hyperplane if classes are linearly separable
- use pseudoinverse if $\mathbf{Y}^t \mathbf{Y}$ is not singular and not too large
- use gradient descent (Widrow-Hoff procedure) otherwise

Support Vector Machines

Problem Definition

Consider a training set of n iid samples

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$$

where \mathbf{x}_i is a vector of length m and

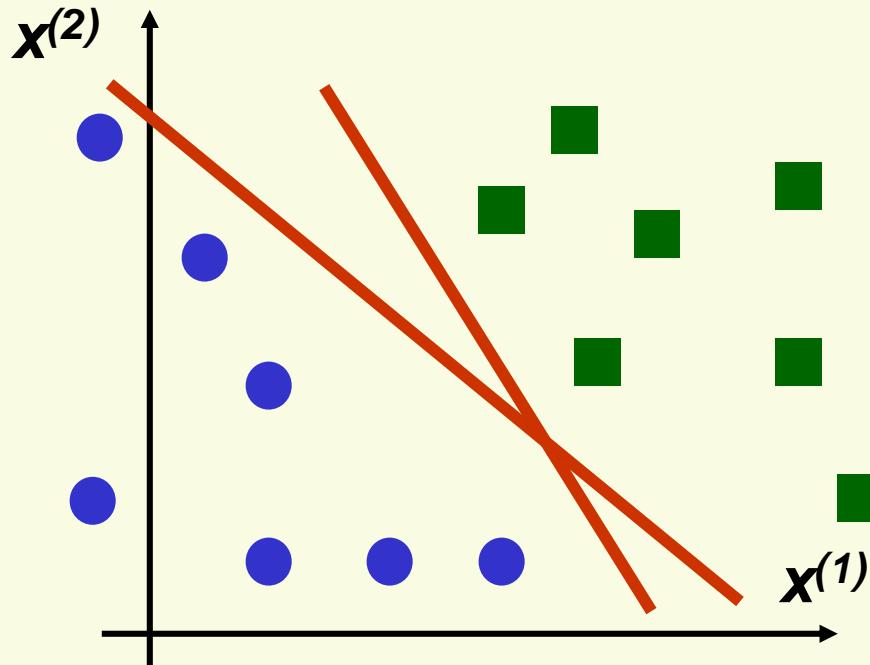
$y_i \in \{+1, -1\}$ is the class label for data point \mathbf{x}_i .

Find a separating hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$

corresponding to the decision function

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

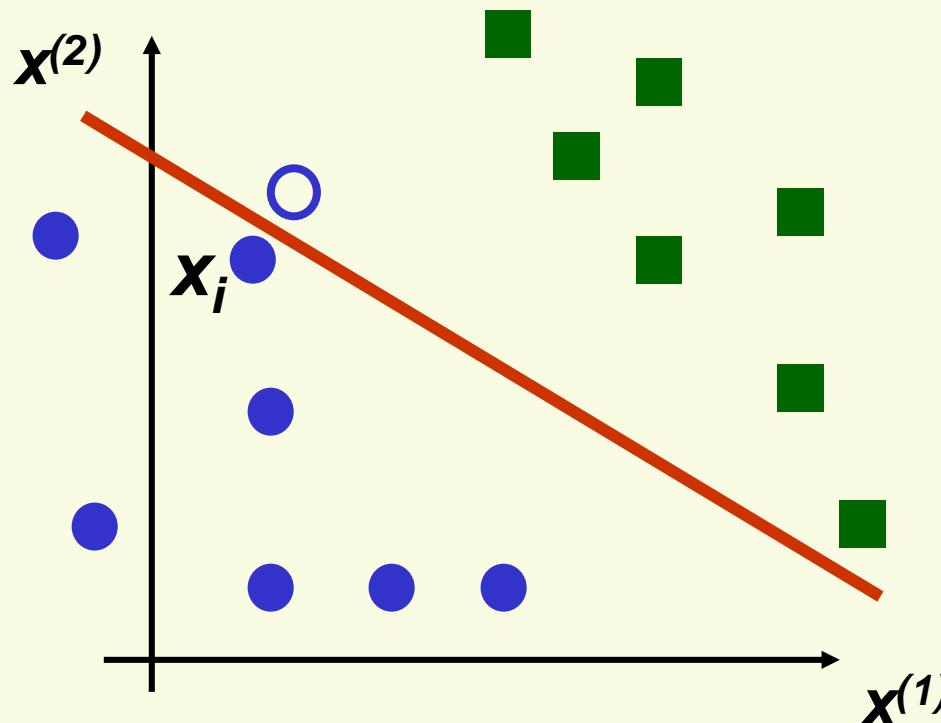
Separating Hyperplanes



- which separating hyperplane should we choose?

Separating Hyperplanes

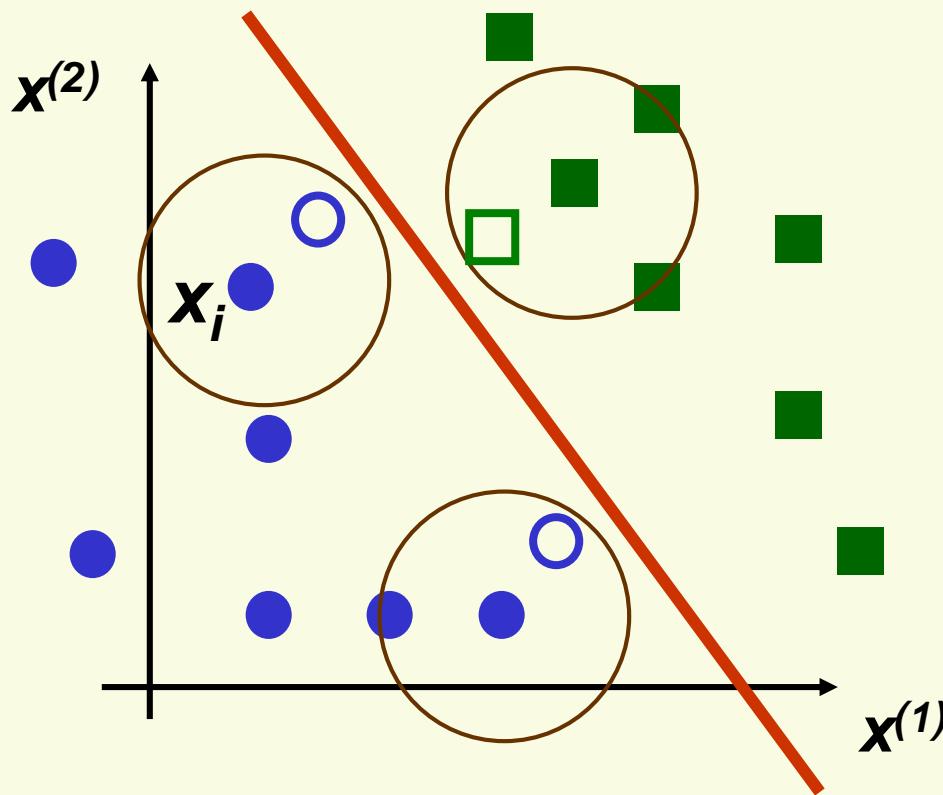
- Training data is just a subset of all possible data
- Suppose hyperplane is close to sample x_i
- If we see new sample close to sample i , it is likely to be on the wrong side of the hyperplane



- Poor generalization (performance on unseen data)

Separating Hyperplanes

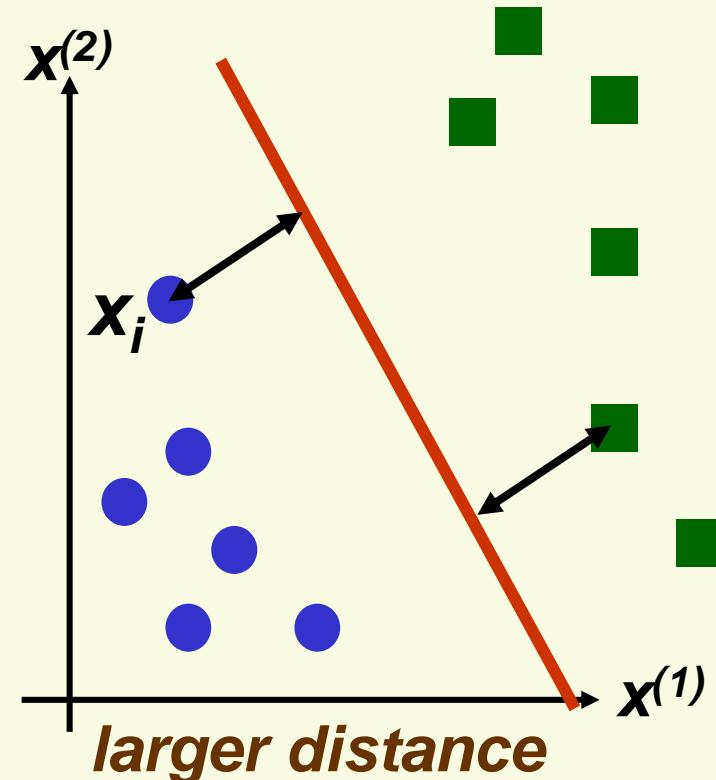
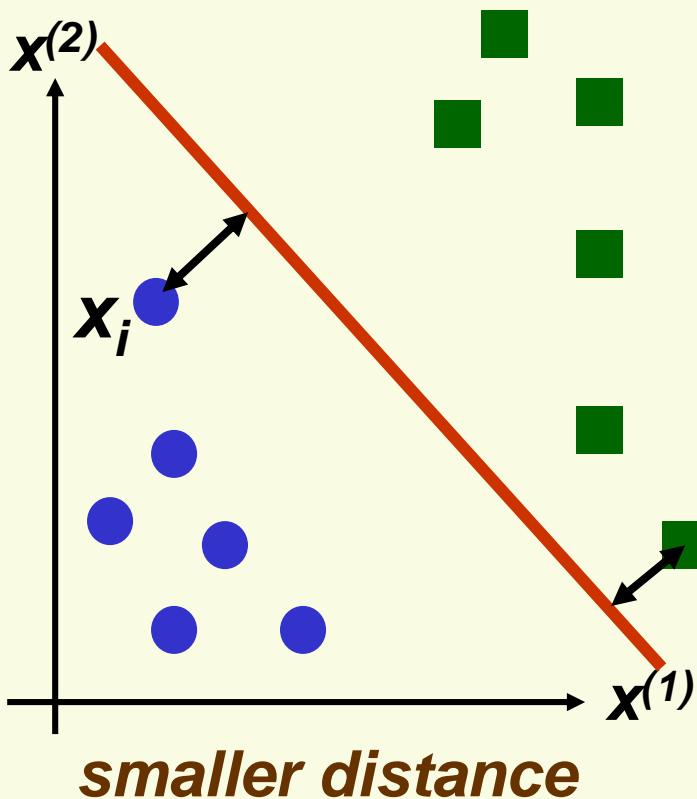
- Hyperplane as far as possible from any sample



- New samples close to the old samples will be classified correctly
- Good generalization

SVM

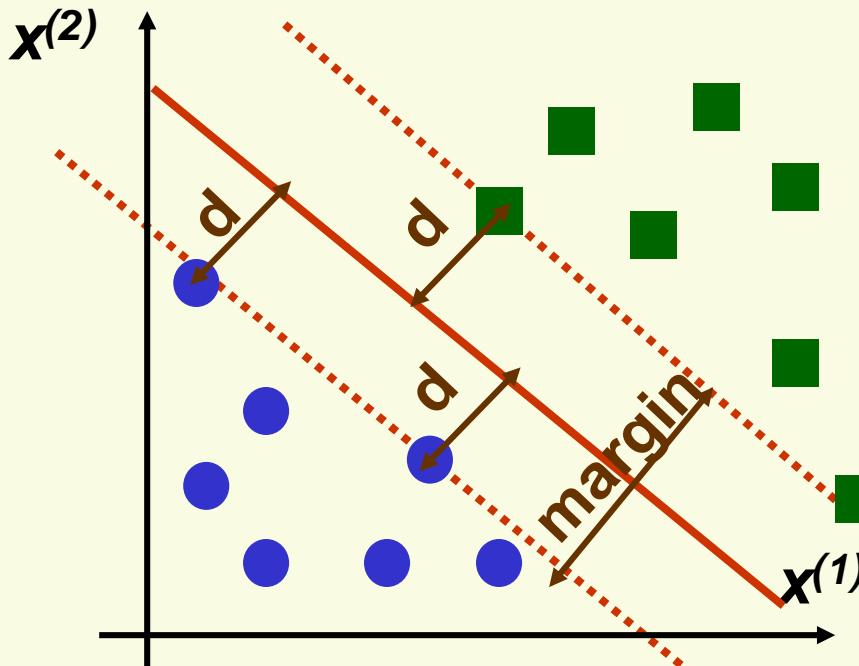
- Idea: maximize distance to the closest example



- For the optimal hyperplane
 - distance to the closest negative example = distance to the closest positive example

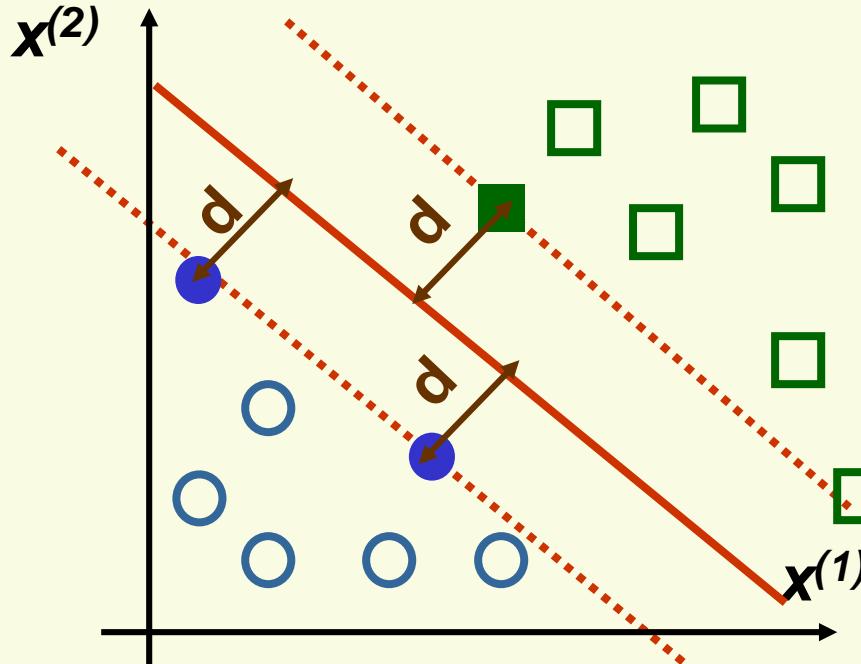
SVM: Linearly Separable Case

- SVM: maximize the *margin*



- **margin** is twice the absolute value of distance **d** of the closest examples to the separating hyperplane
- Better generalization (performance on test data)
 - in practice
 - and in theory

SVM: Linearly Separable Case

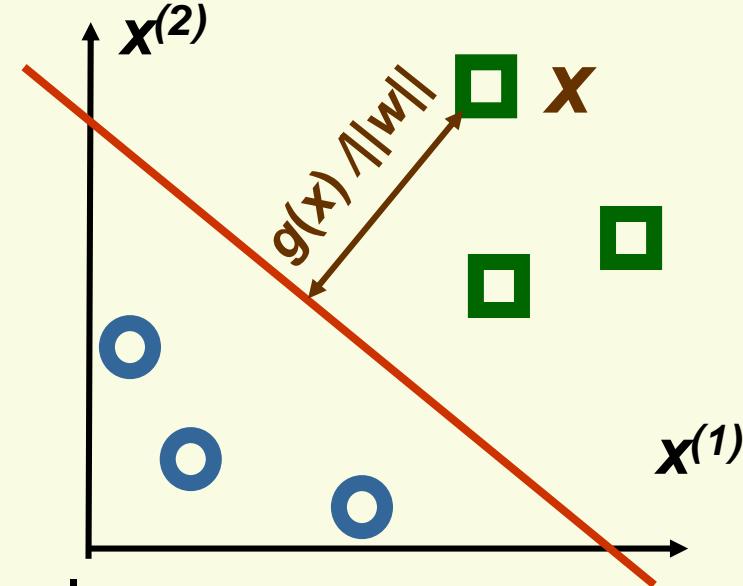


- ***Support vectors*** are the samples closest to the separating hyperplane
 - they are the most difficult patterns to classify
 - Optimal hyperplane is completely defined by support vectors
 - of course, we do not know which samples are support vectors without finding the optimal hyperplane

SVM: Formula for the Margin

- $g(\mathbf{x}) = \mathbf{w}^t \mathbf{x} + b$
- absolute distance between \mathbf{x} and the boundary $g(\mathbf{x}) = 0$

$$\frac{|\mathbf{w}^t \mathbf{x} + b|}{\|\mathbf{w}\|}$$



- distance is unchanged for hyperplane

$$g_1(\mathbf{x}) = \alpha g(\mathbf{x})$$

$$\frac{|\alpha \mathbf{w}^t \mathbf{x} + \alpha b|}{\|\alpha \mathbf{w}\|} = \frac{|\mathbf{w}^t \mathbf{x} + b|}{\|\mathbf{w}\|}$$

- Let \mathbf{x}_i be an example closest to the boundary. Set

$$|\mathbf{w}^t \mathbf{x}_i + b| = 1$$

- Now the largest margin hyperplane is unique

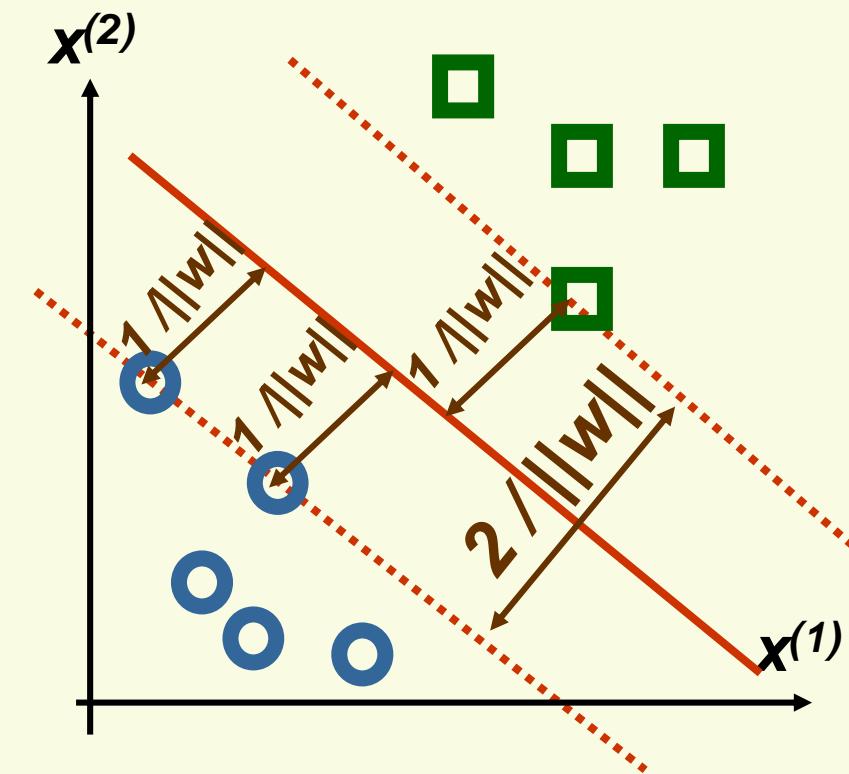
SVM: Formula for the Margin

- For uniqueness, set $|w^t x_i + b| = 1$ for any example x_i closest to the boundary
- now distance from closest sample x_i to $g(x) = 0$ is

$$\frac{|w^t x_i + b|}{\|w\|} = \frac{1}{\|w\|}$$

- Thus the margin is

$$m = \frac{2}{\|w\|}$$



SVM: Optimal Hyperplane

- Maximize margin $m = \frac{2}{\|w\|}$

subject to constraints

$$\begin{cases} w^t x_i + b \geq 1 & y_i = 1 \\ w^t x_i + b \leq -1 & y_i = -1 \end{cases}$$

- Can convert our problem to minimize
 $J(w) = \frac{1}{2} \|w\|^2$ s.t. $y_i(w \cdot x_i + b) \geq 1$

- $J(w)$ is a quadratic function, thus there is a single global minimum

Constrained Quadratic Programming

Primal Problem:

$$\text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i$$

- Introduce Lagrange multipliers $\alpha_i \geq 0$ associated with the constraints
- The solution to the primal problem is equivalent to determining the saddle point of the function

$$L_P \equiv L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{x}_i \cdot \mathbf{w} + b) - 1)$$

Solving Constrained QP

- At saddle point, L_P has minimum requiring

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i = \mathbf{0} \Rightarrow \mathbf{w} = \sum_i \alpha_i \mathbf{y}_i \mathbf{x}_i$$

$$\frac{\partial L_P}{\partial \mathbf{b}} = \sum_i \alpha_i \mathbf{y}_i = \mathbf{0}$$

Primal-Dual

Primal:

$$L_P = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{w} + b) + \sum_{i=1}^n \alpha_i$$

minimize L_P with respect to \mathbf{w}, b ,
subject to $\alpha_i \geq 0$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad \sum_i \alpha_i y_i = 0 \quad \text{substitute}$$

Dual:

$$L_D = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$

maximize L_D with respect to α
subject to $\alpha_i \geq 0, \quad \sum_i \alpha_i y_i = 0$

Solving QP using dual problem

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^t \mathbf{x}_j$$

constrained to $\alpha_i \geq 0 \quad \forall i \quad \text{and} \quad \sum_{i=1}^n \alpha_i y_i = 0$

- $\alpha = \{\alpha_1, \dots, \alpha_n\}$ are new variables, one for each sample
- $L_D(\alpha)$ can be optimized by quadratic programming
- $L_D(\alpha)$ formulated in terms of α
 - it depends on w and b indirectly
- $L_D(\alpha)$ depends on the number of samples, not on dimension of samples

Threshold

- b can be determined from the optimal α and Karush-Kuhn-Tucker (KKT) conditions

$$\alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0, \quad \forall i$$

- $\alpha_i > 0$ implies

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1 \Rightarrow \mathbf{w} \cdot \mathbf{x}_i + b = y_i$$

$$b = y_i - \mathbf{w} \cdot \mathbf{x}_i$$

Support Vectors

- For every sample i , one of the following must hold
 - $\alpha_i = 0$
 - $\alpha_i > 0$ and $y_i(\mathbf{w} \cdot \mathbf{x}_i + b - 1) = 0$
- Many $\alpha_i = 0 \Rightarrow \mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$ sparse solution
- Samples with $\alpha_i > 0$ are **Support Vectors** and they are the closest to the separating hyperplane
- Optimal hyperplane is completely defined by support vectors

SVM: Classification

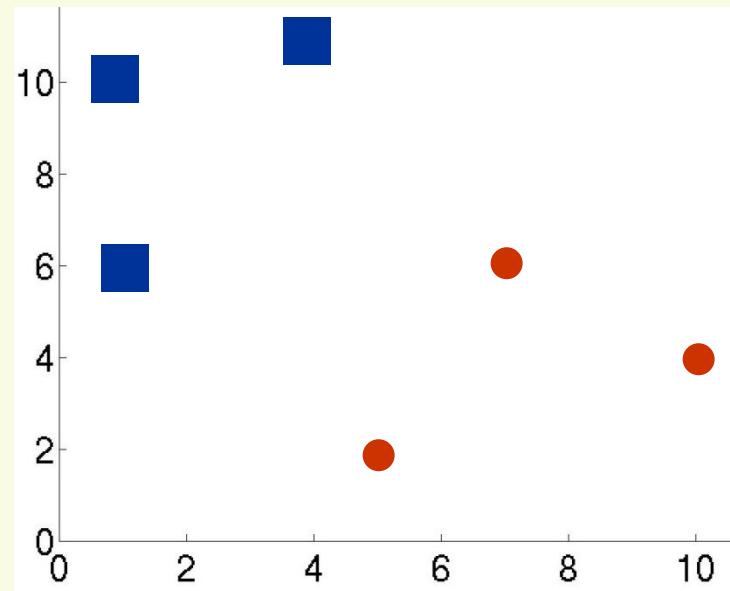
- Given a new sample x , finds its label y

$$y = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$$

SVM: Example

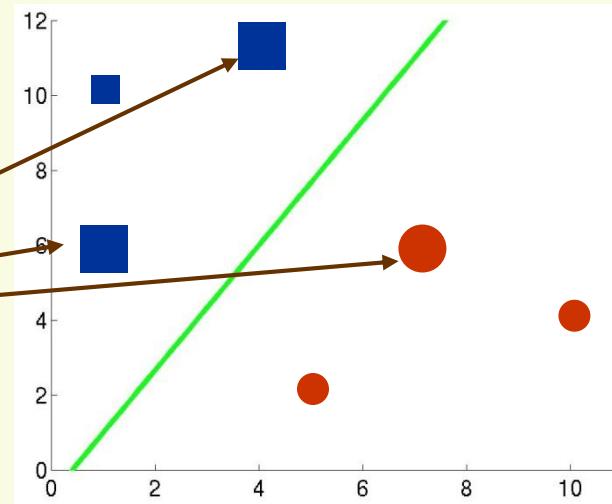
- Class 1: [1,6], [1,10], [4,11]
- Class 2: [5,2], [7,6], [10,4]



SVM: Example

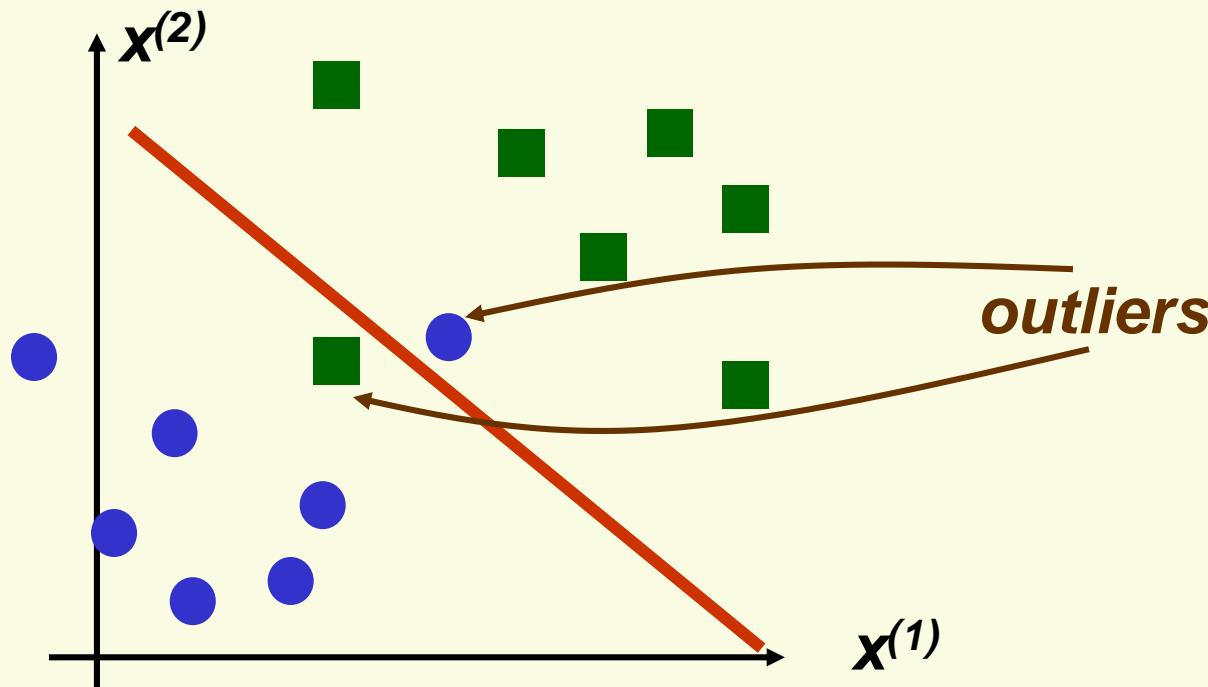
- Solution $\alpha = \begin{bmatrix} 0.036 \\ 0 \\ 0.039 \\ 0 \\ 0.076 \\ 0 \end{bmatrix}$ *support vectors*
- find w using $w = \sum_{i=1}^n \alpha_i y_i x_i = (\alpha \cdot y)^t x = \begin{bmatrix} -0.33 \\ 0.20 \end{bmatrix}$
- since $\alpha_1 > 0$, can find b using

$$b = y_1 - w^t x_1 = 0.13$$



SVM: Non Separable Case

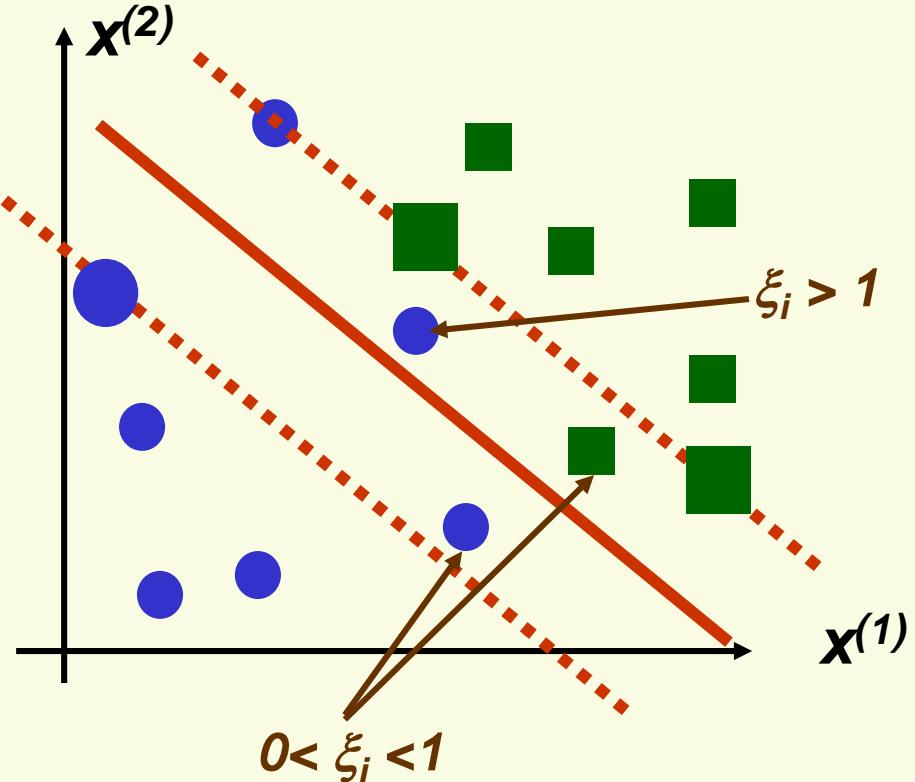
- Data is most likely to be not linearly separable, but linear classifier may still be appropriate



- Can apply SVM in non linearly separable case
 - data should be “almost” linearly separable for good performance

SVM with slacks

- Use nonnegative “slack” variables ξ_1, \dots, ξ_n (one for each sample)
- Change constraints from $y_i(\mathbf{w}^t \mathbf{x}_i + b) \geq 1 \quad \forall i$ to
$$y_i(\mathbf{w}^t \mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i$$
- ξ_i is a measure of deviation from the ideal position for sample i
 - $\xi_i > 1$ sample i is on the wrong side of the separating hyperplane
 - $0 < \xi_i < 1$ sample i is on the right side of separating hyperplane but within the region of maximum margin



SVM with slacks

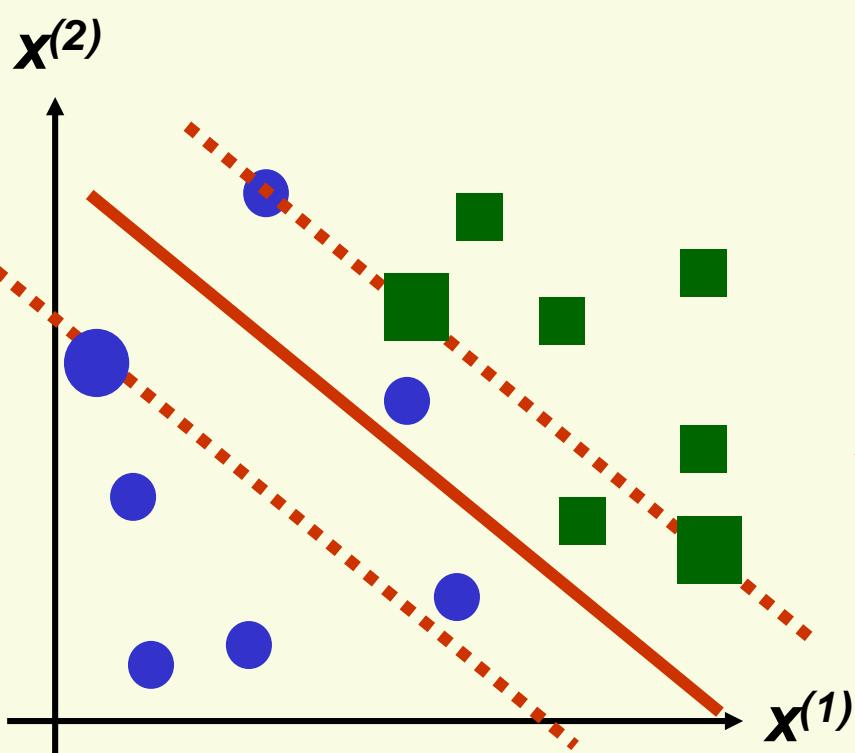
- Would like to minimize

$$J(\mathbf{w}, \xi_1, \dots, \xi_n) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

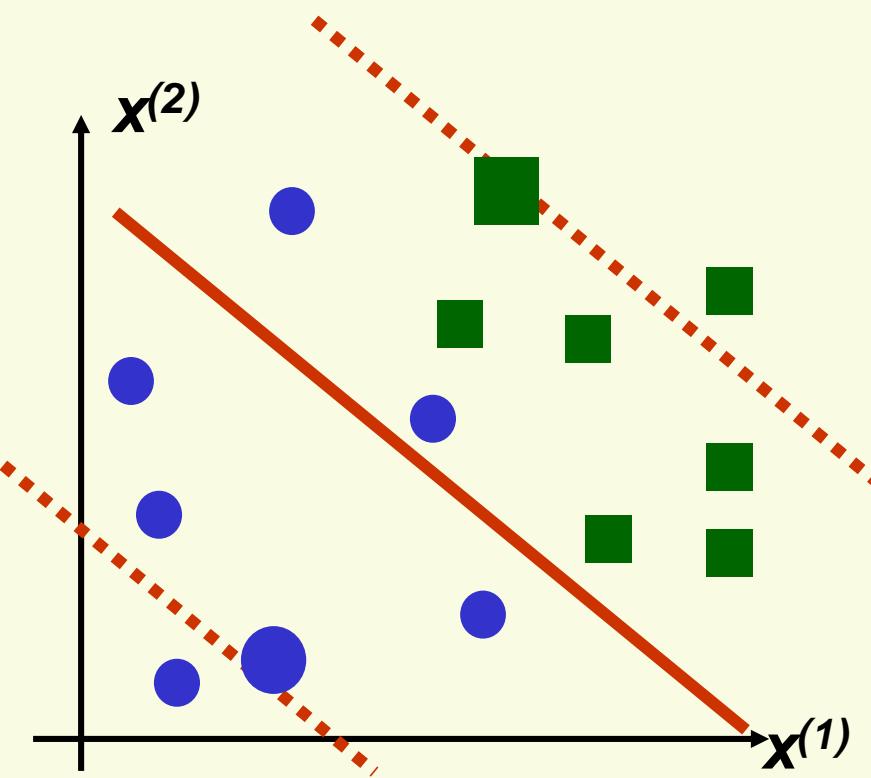
- constrained to $y_i(\mathbf{w}^t \mathbf{x}_i + b) \geq 1 - \xi_i$ and $\xi_i \geq 0 \quad \forall i$
- $C > 0$ is a constant which measures relative weight of the first and second terms
 - if C is small, we allow a lot of samples not in ideal position
 - if C is large, we want to have very few samples not in ideal position

SVM with slacks

$$J(w, \xi_1, \dots, \xi_n) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$$



large C , few samples not in ideal position



small C , a lot of samples not in ideal position

SVM with slacks– Dual Formulation

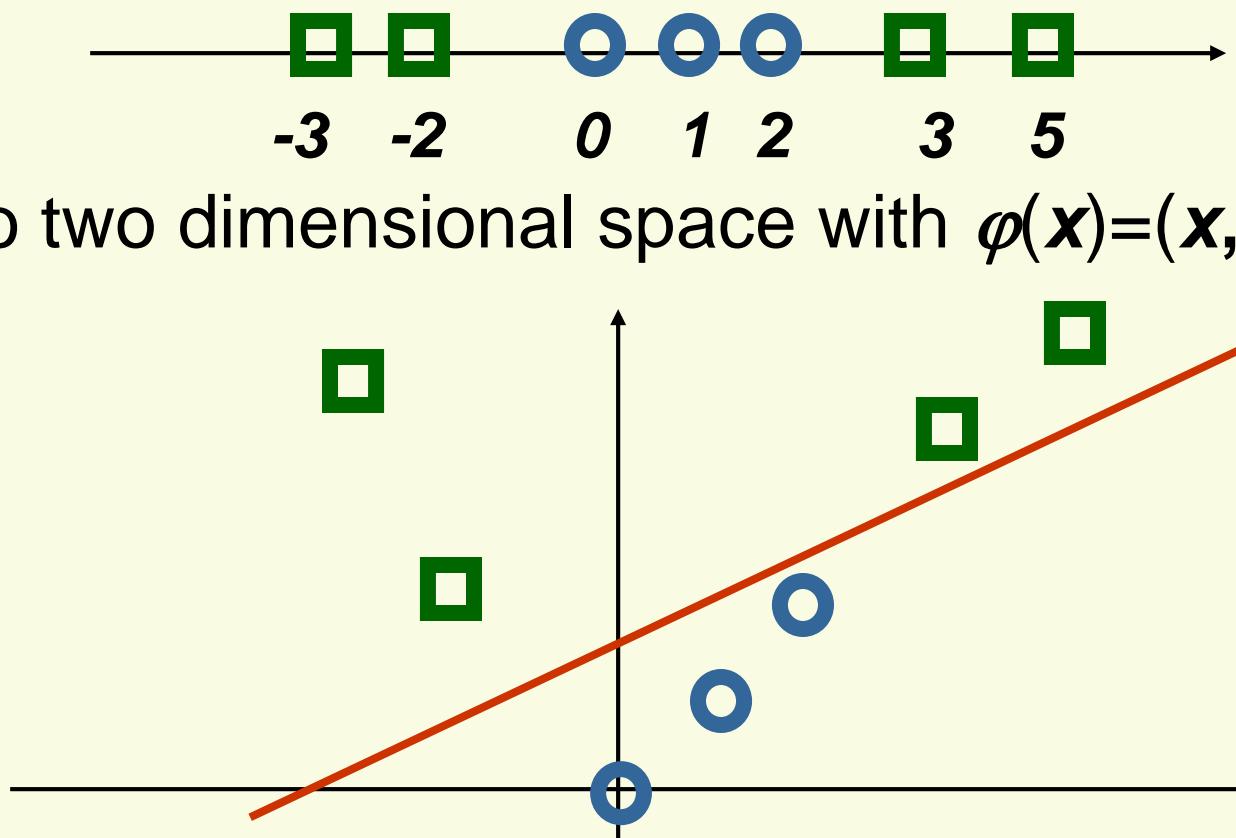
$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^t \mathbf{x}_j$$

$$\text{constrained to } 0 \leq \alpha_i \leq C \quad \forall i \quad \text{and} \quad \sum_{i=1}^n \alpha_i y_i = 0$$

- find \mathbf{w} using $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$
- solve for \mathbf{b} using any $0 < \alpha_i < C$ and $\alpha_i [y_i (\mathbf{w}^t \mathbf{x}_i + \mathbf{b}) - 1] = 0$

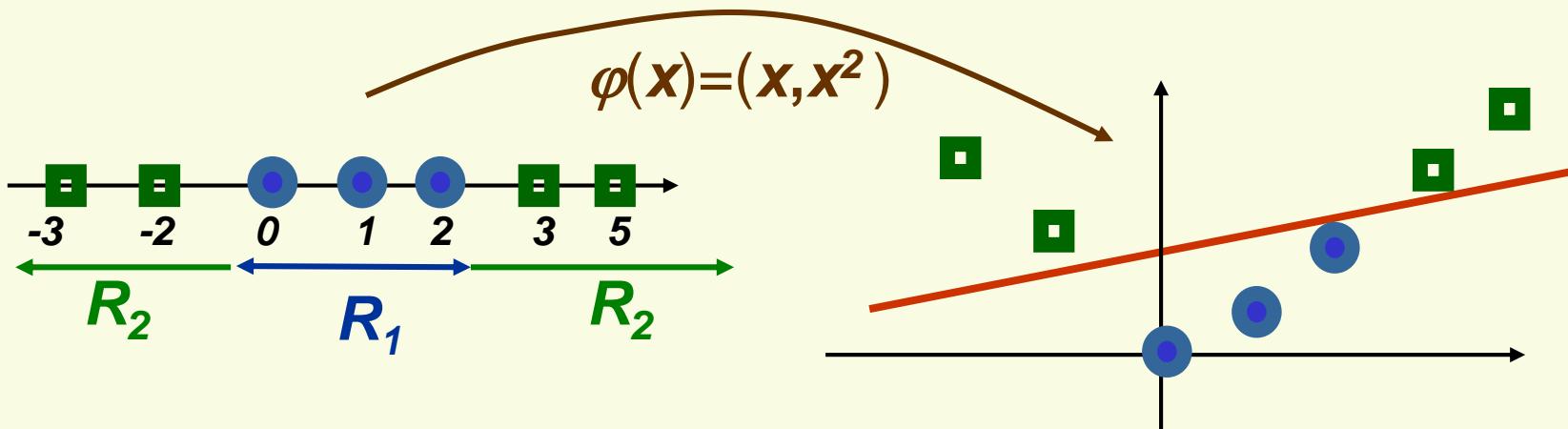
Non Linear Mapping

- Cover's theorem:
 - “pattern-classification problem cast in a high dimensional space non-linearly is more likely to be linearly separable than in a low-dimensional space”
- One dimensional space, not linearly separable
- Lift to two dimensional space with $\varphi(\mathbf{x})=(\mathbf{x}, \mathbf{x}^2)$



Non Linear Mapping

- Solve a non linear classification problem with a linear classifier
 - Project data \mathbf{x} to high dimension using function $\phi(\mathbf{x})$
 - Find a linear discriminant function for transformed data $\phi(\mathbf{x})$
 - Final nonlinear discriminant function is $g(\mathbf{x}) = \mathbf{w}^t \phi(\mathbf{x}) + w_0$

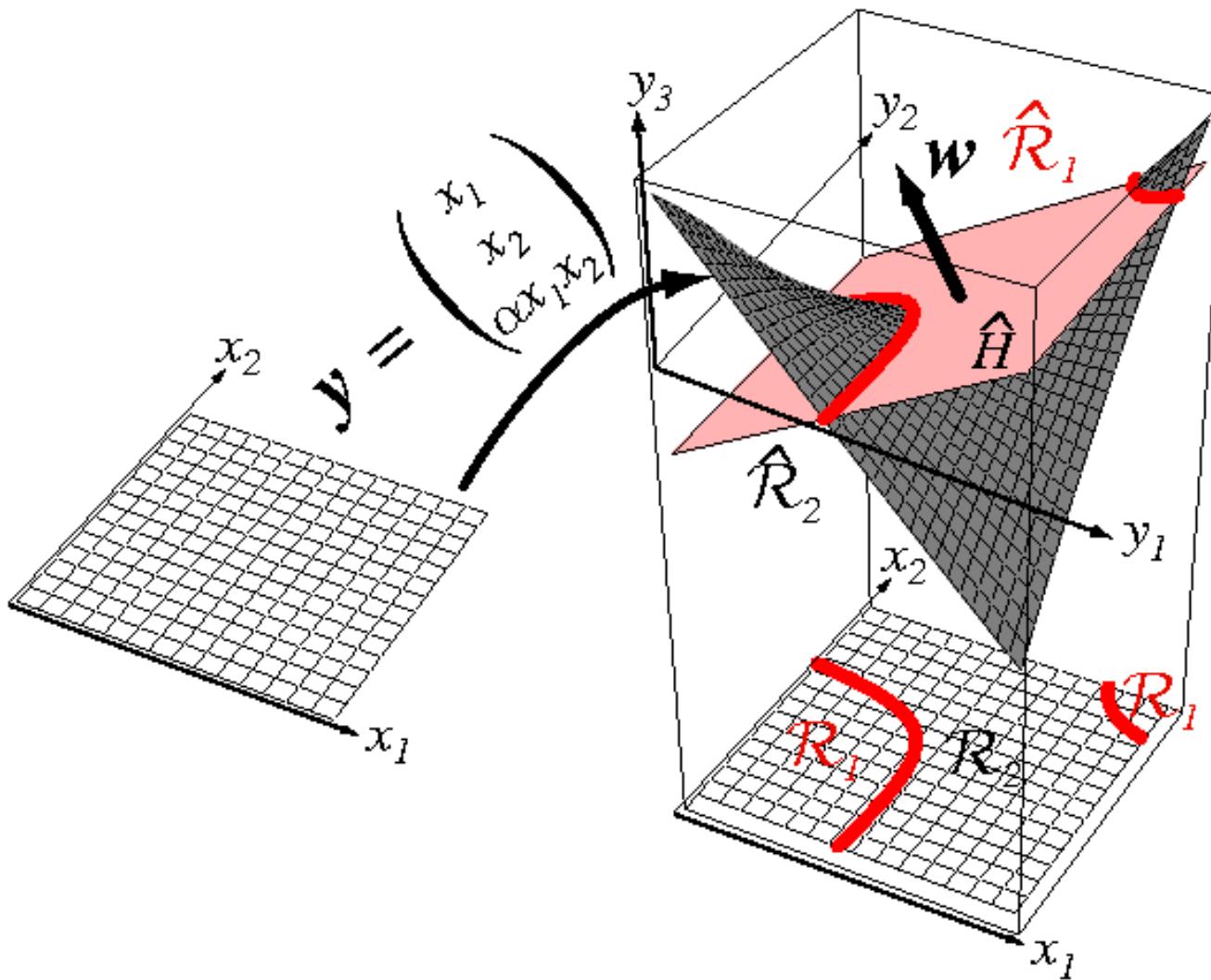


- In 2D, discriminant function is linear

$$g\left(\begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{bmatrix}\right) = [\mathbf{w}_1 \quad \mathbf{w}_2] \begin{bmatrix} \mathbf{x}^{(1)} \\ \mathbf{x}^{(2)} \end{bmatrix} + w_0$$

- In 1D, discriminant function is not linear $g(\mathbf{x}) = w_1 \mathbf{x} + w_2 \mathbf{x}^2 + w_0$

Non Linear Mapping: Another Example



Non Linear SVM

- Can use any linear classifier after lifting data into a higher dimensional space. However we will have to deal with the “curse of dimensionality”
 1. poor generalization to test data
 2. computationally expensive
- SVM handles the “curse of dimensionality” problem:
 1. enforcing largest margin permits good generalization
 - It can be shown that generalization in SVM is a function of the margin, independent of the dimensionality
 2. computation in the higher dimensional case is performed only implicitly through the use of ***kernel*** functions

Non Linear SVM: Kernels

- Recall SVM optimization

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^t \mathbf{x}_j$$

and classification $y = \text{sign}(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b)$

- Note that samples \mathbf{x}_i appear only through the dot products $\mathbf{x}_i^t \mathbf{x}_j$, $\mathbf{x}_i^t \mathbf{x}$.
- If we lift \mathbf{x}_i to high dimensional space F using $\varphi(\mathbf{x})$, need to compute high dimensional product $\varphi(\mathbf{x}_i)^t \varphi(\mathbf{x}_j)$

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \varphi(\mathbf{x}_i)^t \varphi(\mathbf{x}_j)$$

- The dimensionality of space F not necessarily important. May not even know the map φ .

Kernel

- A function that returns the value of the dot product between the images of the two arguments:

$$K(x,y) = \varphi(x)^t \varphi(y)$$

- Given a function K, it is possible to verify that it is a kernel.

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \varphi(x_i)^t \varphi(x_j)$$

$K(x_i, x_j)$

- Now we only need to compute $K(x_i, x_j)$ instead of $\varphi(x_i)^t \varphi(x_j)$
- “kernel trick”: do not need to perform operations in high dimensional space explicitly

Kernel Matrix

- (aka the Gram matrix):

$K =$

$K(1,1)$	$K(1,2)$	$K(1,3)$...	$K(1,m)$
$K(2,1)$	$K(2,2)$	$K(2,3)$...	$K(2,m)$
...
$K(m,1)$	$K(m,2)$	$K(m,3)$...	$K(m,m)$

- The central structure in kernel machines
- Contains all necessary information for the learning algorithm
- Fuses information about the data AND the kernel
- Many interesting properties:

Mercer's Theorem

- The kernel matrix is Symmetric Positive Definite
- Any symmetric positive definite matrix can be regarded as a kernel matrix, that is as an inner product matrix in some space

Every (semi)positive definite, symmetric function is a kernel: i.e. there exists a mapping φ such that it is possible to write:

$$K(x, y) = \varphi(x)^t \varphi(y)$$

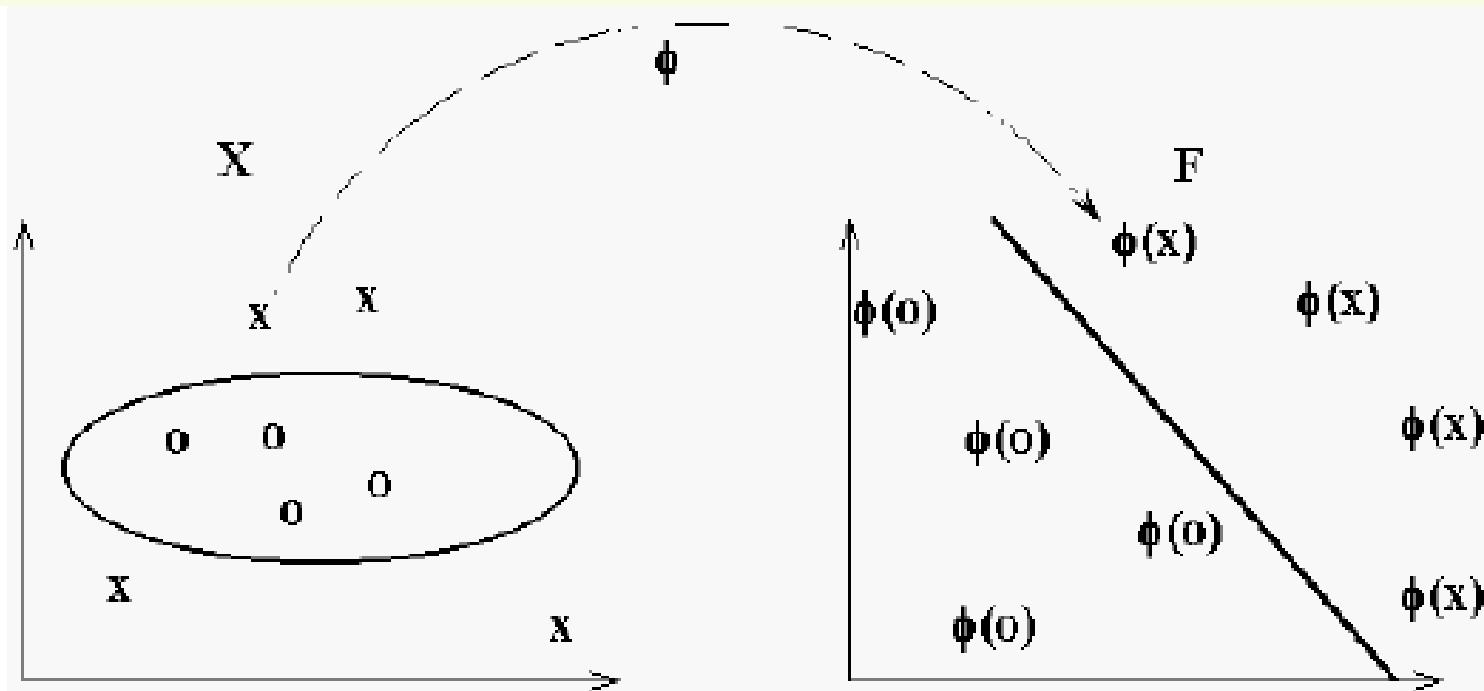
Positive definite $\int_{\forall f \in L_2} K(x, y) f(x) f(y) dx dy \geq 0$

Examples of Kernels

- Some common choices (both satisfying Mercer's condition):
 - Polynomial kernel $K(x_i, x_j) = (x_i^T x_j + 1)^p$
 - Gaussian radial Basis kernel (data is lifted in infinite dimension)

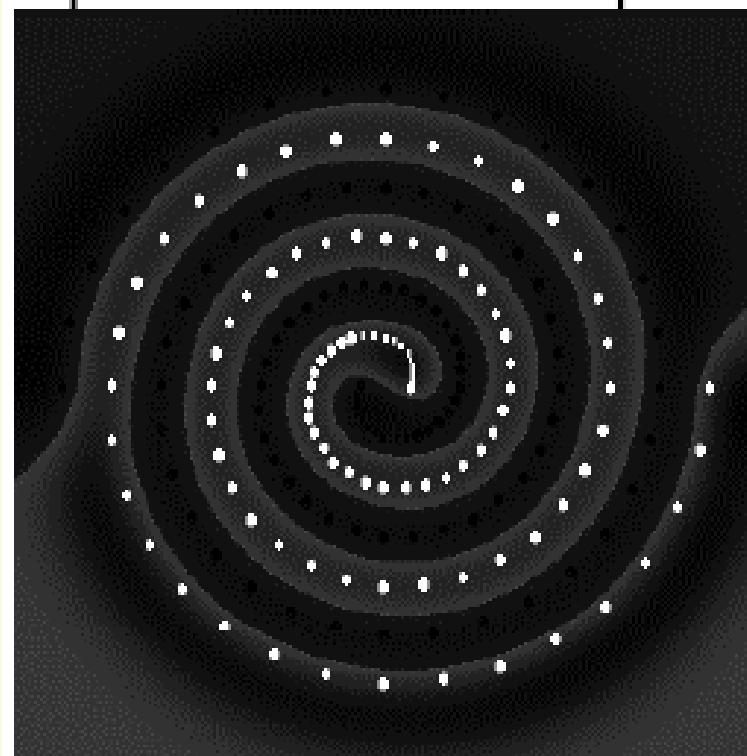
$$K(x_i, x_j) = \exp\left(-\frac{1}{2\sigma^2} \|x_i - x_j\|^2\right)$$

Example Polynomial Kernels



Example: the two spirals

- Separated by a hyperplane in feature space
(gaussian kernels)



Making Kernels

- The set of kernels is closed under some operations. If K, K' are kernels, then:
- $K+K'$ is a kernel
- cK is a kernel, if $c>0$
- $aK+bK'$ is a kernel, for $a,b >0$
- Etc etc etc.....
- can make complex kernels from simple ones: modularity !

Non Linear SVM Recipe

- Start with data $\mathbf{x}_1, \dots, \mathbf{x}_n$ which lives in feature space of dimension d
- Choose kernel $K(\mathbf{x}_i, \mathbf{x}_j)$ corresponding to some function $\varphi(\mathbf{x}_i)$ which takes sample \mathbf{x}_i to a higher dimensional space
- Find the largest margin linear discriminant function in the higher dimensional space by using quadratic programming package to solve:

$$\text{maximize } L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

constrained to $0 \leq \alpha_i \leq C \quad \forall i$ and $\sum_{i=1}^n \alpha_i y_i = 0$

Non Linear SVM Recipe

- Weight vector w in the high dimensional space:
$$w = \sum_i^n \alpha_i y_i \varphi(x_i)$$
- Linear discriminant function of largest margin in the high dimensional space:
$$g(\varphi(x)) = w^t \varphi(x) = \left(\sum_{x_i \in S} \alpha_i y_i \varphi(x_i) \right)^t \varphi(x)$$
- Non linear discriminant function in the original space:
$$g(x) = \left(\sum_{x_i \in S} \alpha_i y_i \varphi(x_i) \right)^t \varphi(x) = \sum_{x_i \in S} \alpha_i y_i \varphi^t(x_i) \varphi(x) = \sum_{x_i \in S} \alpha_i y_i K(x_i, x)$$
- decide class 1 if $g(x) > 0$, otherwise decide class 2

Non Linear SVM

- Nonlinear discriminant function

$$g(\mathbf{x}) = \sum_{\mathbf{x}_i \in S} \alpha_i z_i K(\mathbf{x}_i, \mathbf{x})$$

$$g(\mathbf{x}) = \sum \text{weight of support vector } \mathbf{x}_i \quad \mp 1 \quad \text{"inverse distance" from } \mathbf{x} \text{ to support vector } \mathbf{x}_i$$

most important training samples,
i.e. support vectors

$$K(\mathbf{x}_i, \mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{x}_i - \mathbf{x}\|^2\right)$$

SVM Summary

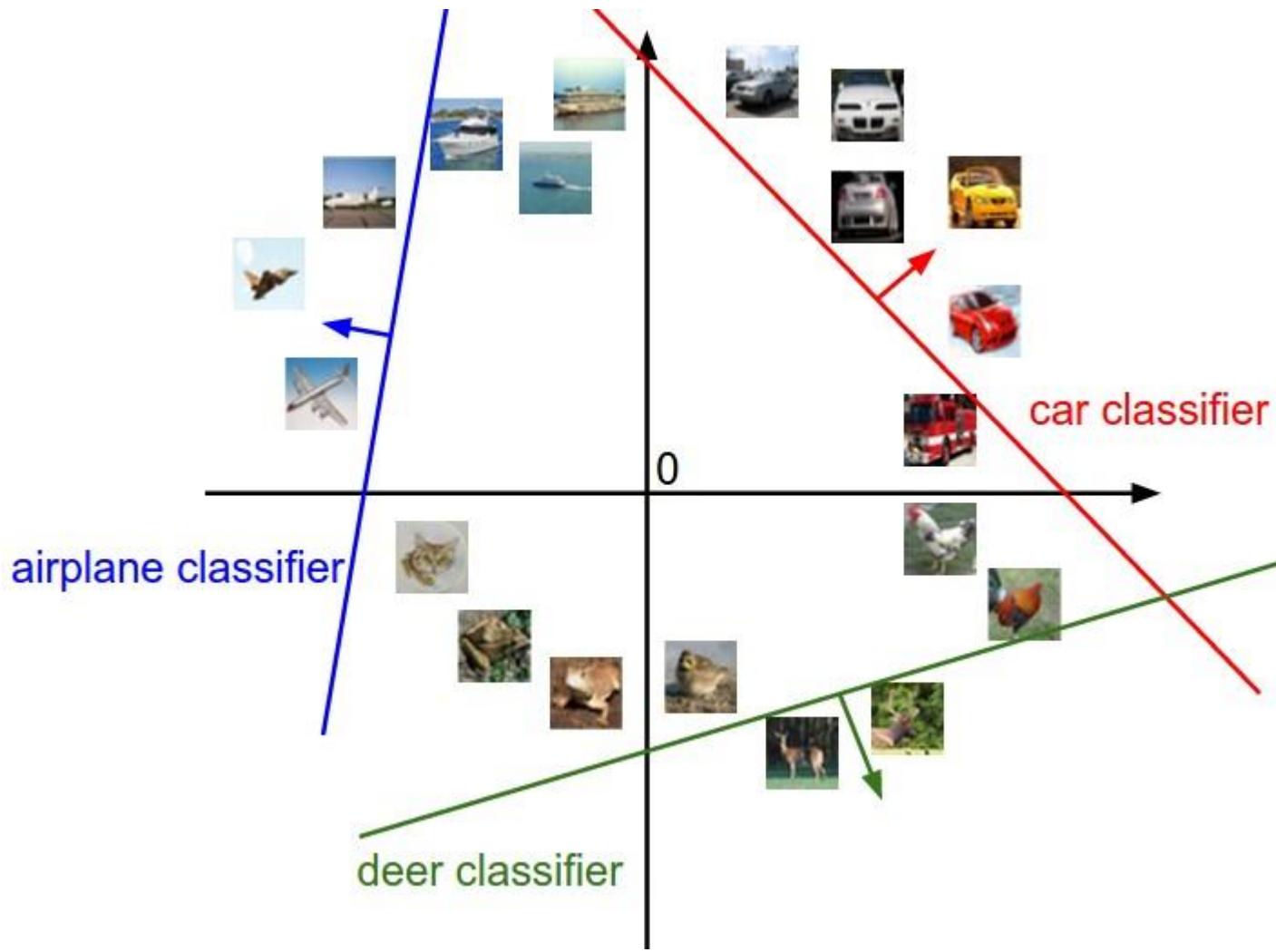
- Advantages:
 - Based on nice theory
 - excellent generalization properties
 - objective function has no local minima
 - can be used to find non linear discriminant functions
 - Complexity of the classifier is characterized by the number of support vectors rather than the dimensionality of the transformed space
- Disadvantages:
 - It's not clear how to select a kernel function in a principled manner
 - tends to be slower than other methods (in non-linear case).

Multiclass Classification

Based on the Stanford course “CNN for visual recognition”

<http://cs231n.github.io/linear-classify/#softmax>

Example



Loss Function

Training data: $\{x_t, y_t\}, t = 1, \dots, n; x_t \in R^d, y_t \in \{1, \dots, C\}$

Prediction: $f(x_t, \theta)$, θ - learned parameters

How do we compare the prediction $f(x_t, \theta)$ with the true label y_t ?

Loss: $L(f(x_t, \theta), y_t)$

Intuitively, the loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well.

MSE Loss

- Samlpe loss

$$L_i = (f(x_i) - y_i)^2$$

- Full loss

$$L = \sum_i (f(x_i) - y_i)^2$$

- Have saw this before

Multiclass SVM Loss

- The correct class for each input should have a score higher than the incorrect classes by some fixed margin Δ .
- Assume that the score of the j -th class is $s_j = f(x_i, \Theta)$,
- The Multiclass SVM loss for the i -th example is then formalized as:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Example: suppose that we have three classes and for some x_i we received the scores $s = [13, -7, 11]$ while $y_i = 1$ and that margin is set to $\Delta=10$.

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

0 as incorrect class score (-7) is smaller than the correct class score (13) by at least margin 10

8, since the difference between the correct class score and the incorrect one is smaller than the margin

Multiclass Linear SVM Loss

$$\theta \triangleq W, \quad s_j = f(x_i, W)_j$$

$$f(x_i, W) = Wx_i$$

$$W = \begin{bmatrix} w_1 \\ \dots \\ w_c \end{bmatrix} \quad w_j \triangleq \begin{bmatrix} w_j \\ b_j \end{bmatrix} \quad x \triangleq \begin{bmatrix} x \\ 1 \end{bmatrix}$$

$$L_i = \sum_{j \neq y_i} \max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

Hinge Loss: $\max(0, -)$

Regularization

- Suppose that we have a dataset and a set of parameters \mathbf{W} that correctly classify every example and $L_i = 0$ for all i .
- For any $k > 0$, $k\mathbf{W}$ will also produce zero loss.
- We wish to encode some preference for a certain set of weights \mathbf{W} over others to remove this ambiguity.
- Extend the loss with a regularization penalty

$$R(\mathbf{W}) = \sum_k \sum_l W_{k,l}^2$$

$$L = \frac{1}{N} \sum_i L_i + \lambda R(\mathbf{W})$$

The regularization forces large margin. The formulation is equivalent to the one we saw earlier for the binary case.

Multiclass Linear SVM Loss

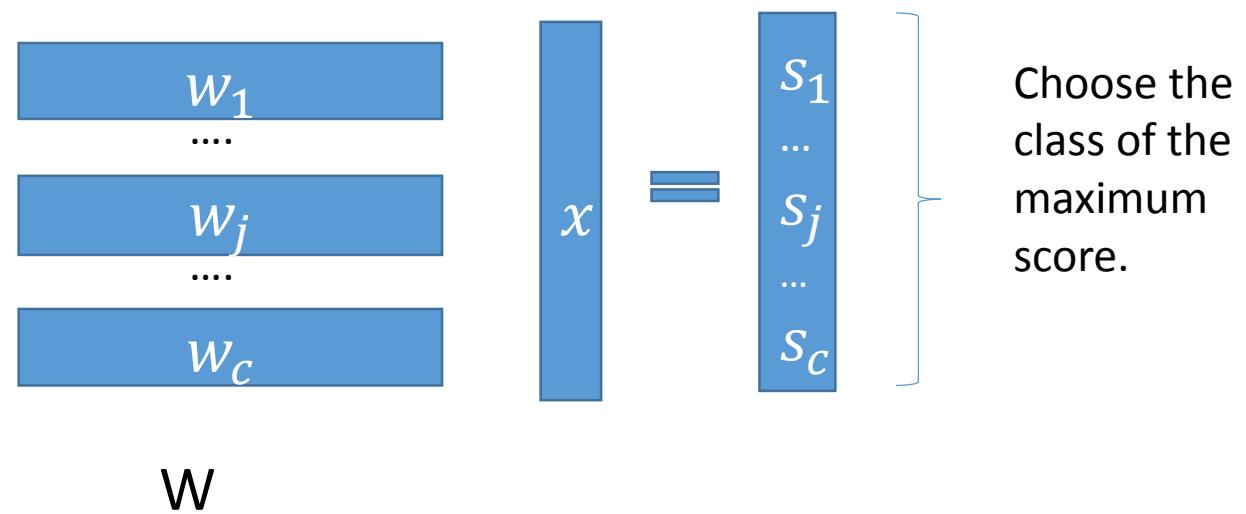
$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

- N is the number of training examples
- λ is a regularization parameter (There is no simple way of setting it; usually is determined by cross-validation).
- The parameters λ and Δ control the same tradeoff, thus we can safely set $\Delta=1$.
- The magnitude of the W has direct effect on the scores and their difference: shrink $W \rightarrow$ increases the difference, increase $W \rightarrow$ decreases the difference.
- Therefore, the exact value of the margin between the scores is unimportant. The only real tradeoff is how large we allow the weights to grow (controlled by λ).

Multiclass Linear SVM Prediction

- Given the trained parameters W , we can classify an unseen input x as : $j^* = \operatorname{argmax}_j s_j$

where $s_j = f(x, W)_j$, $f(x, W) = Wx$



One vs All Multiclass SVM

- For each class $j = 1, \dots, C$ train a binary SVM, in which
 - the positive class ($y=1$) contains the training samples of class j
 - the negative class ($y=-1$) includes the samples of all other classes $i \neq j$.
- To classify an unseen input x , compute $s_j = w_j^T x$ for all $j=1, \dots, C$ and predict the class as follows: $j^* = \operatorname{argmax}_j s_j$

All vs All trains binary classifiers for all pairs of classes – thus is computationally expensive and less popular

Softmax Function

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Squashes a vector z to a vector of values between zero and one that sum to one.

Softmax Classifier

- Multiclass SVM treats $f(x_i, W) = Wx_i$ as (uncalibrated and possibly difficult to interpret) scores for each class.
- **Softmax classifier** gives a slightly more intuitive output (normalized class probabilities) and has a probabilistic interpretation.
- The function mapping $f(x_i, W) = Wx_i$ is the same, but we interpret these scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

$$L_i = -\log \left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \right) \quad \text{or equivalently} \quad L_i = -f_{y_i} + \log \sum_j e^{f_j}$$

where f_{y_i} corresponds to the j-th element of the vector of class scores f.

Full Loss:

$$L = \frac{1}{N} \sum_i L_i + \lambda R(W)$$

Information-Theoretic View

- The *cross-entropy* between a “true” distribution p and an estimated distribution q is defined as

$$H(p, q) = - \sum_x p(x) \log q(x)$$

- The Softmax classifier minimizes the cross-entropy between the estimated class probabilities $q = e^{f_{y_i}} / \sum_j e^{f_j}$ and the “true” distribution, which in this interpretation is the distribution where all probability mass is on the correct class: $p = [0, 0, \dots, 1, 0, 0]$ (contains a single 1 at the y_i th position)

Probabilistic Interpretation

$$P(y_i|x_i; W) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

- can be interpreted as the probability assigned to the correct label y_i given the input x_i and parametrized by W .
- In the probabilistic interpretation, we are minimizing the negative log likelihood of the correct class, which can be interpreted as performing *Maximum Likelihood Estimation* (MLE).
- A nice feature of this view is that we can now also interpret the regularization term $R(W)$ in the full loss function as coming from a Gaussian prior over the weight matrix W (we will show the derivation later on) – in that case it's *Maximum a posteriori* (MAP) estimation.

Numeric Stability

- In practice, the intermediate terms $e^{f_{y_i}}$ and $\sum_j e^{f_j}$ may be very large due to the exponentials.
- Dividing large numbers can be numerically unstable, so it is important to use a normalization trick.
- Notice that if we multiply the top and bottom of the fraction by a constant C it doesn't change the result:

$$\frac{Ce^{f_{y_i}}}{C \sum_j e^{f_j}} = \frac{e^{f_{y_i} + \log C}}{\sum_j e^{f_j + \log C}}$$

A common choice for C is to set $\log C = -\max_j f_j$. This simply shifts the values inside the vector f so that the highest value is zero.

Decision Trees

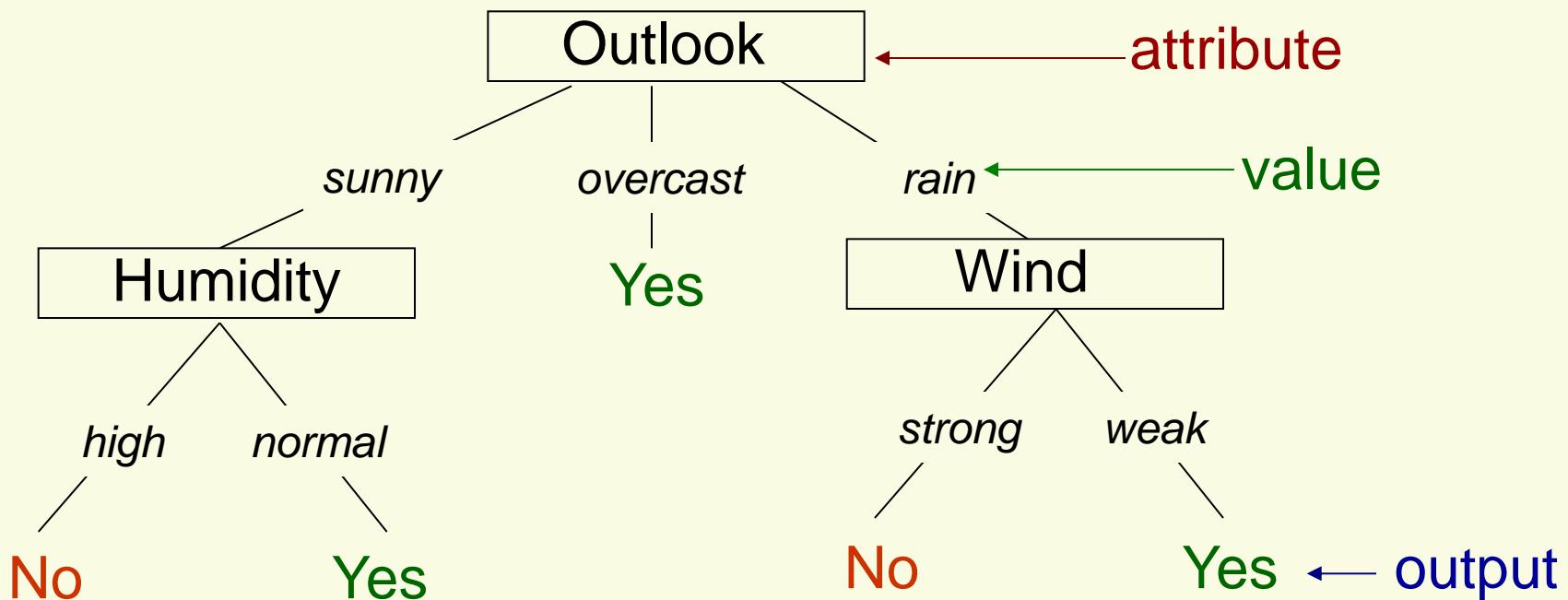
Introduction

- Intuitive to classify a pattern through sequence of questions.
- Next question depends on the answer to the current question.
- Particularly useful for nonmetric data
- The answers could be yes/no, true/false, $\text{property} \in \text{set_of_values}$.

Decision Tree Representation

- Decision trees classify **instances** by sorting them down the tree from the **root node** to some **leaf node**, which provides the classification of the instance.
- Each node in the tree specifies a **test** of some **attribute** of the instance, and each **branch** descending from that node corresponds to one of the possible **values** for this attribute.

Decision Tree Representation

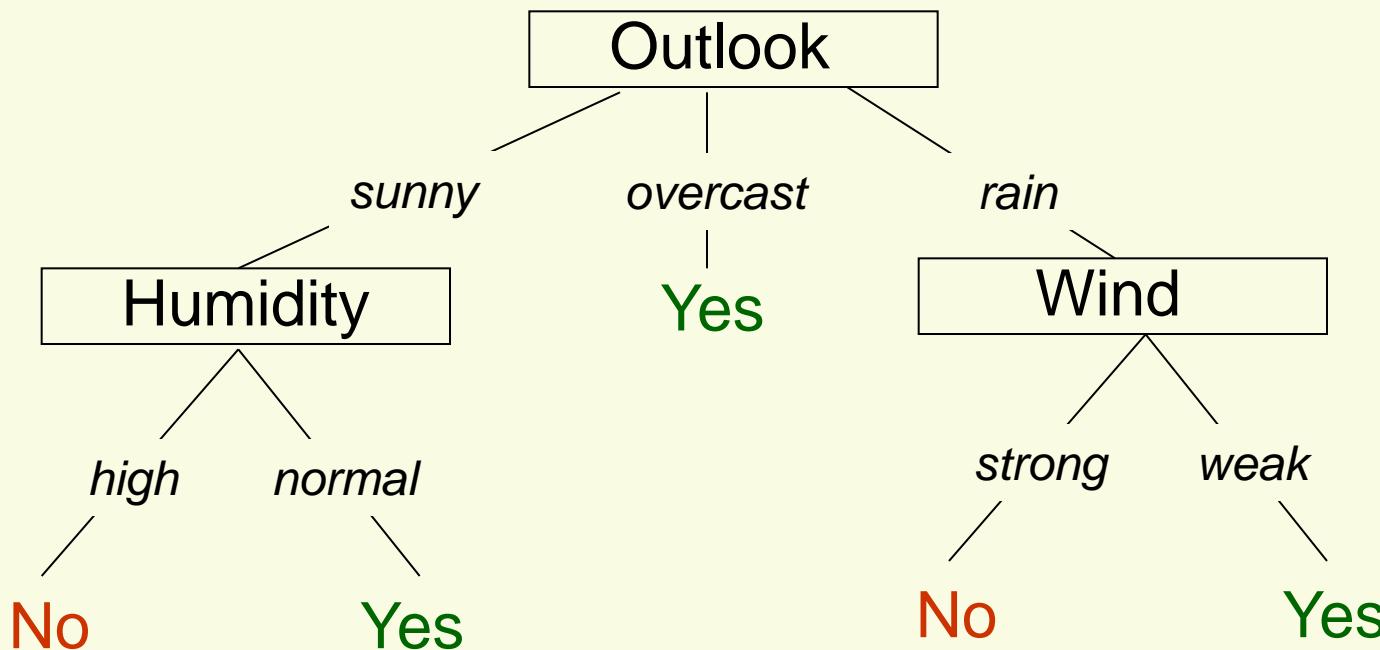


Each internal node: test one attribute X_i

Each branch from a node: selects one value for X_i

Each leaf node: predicts Y

Decision Tree Representation: Example



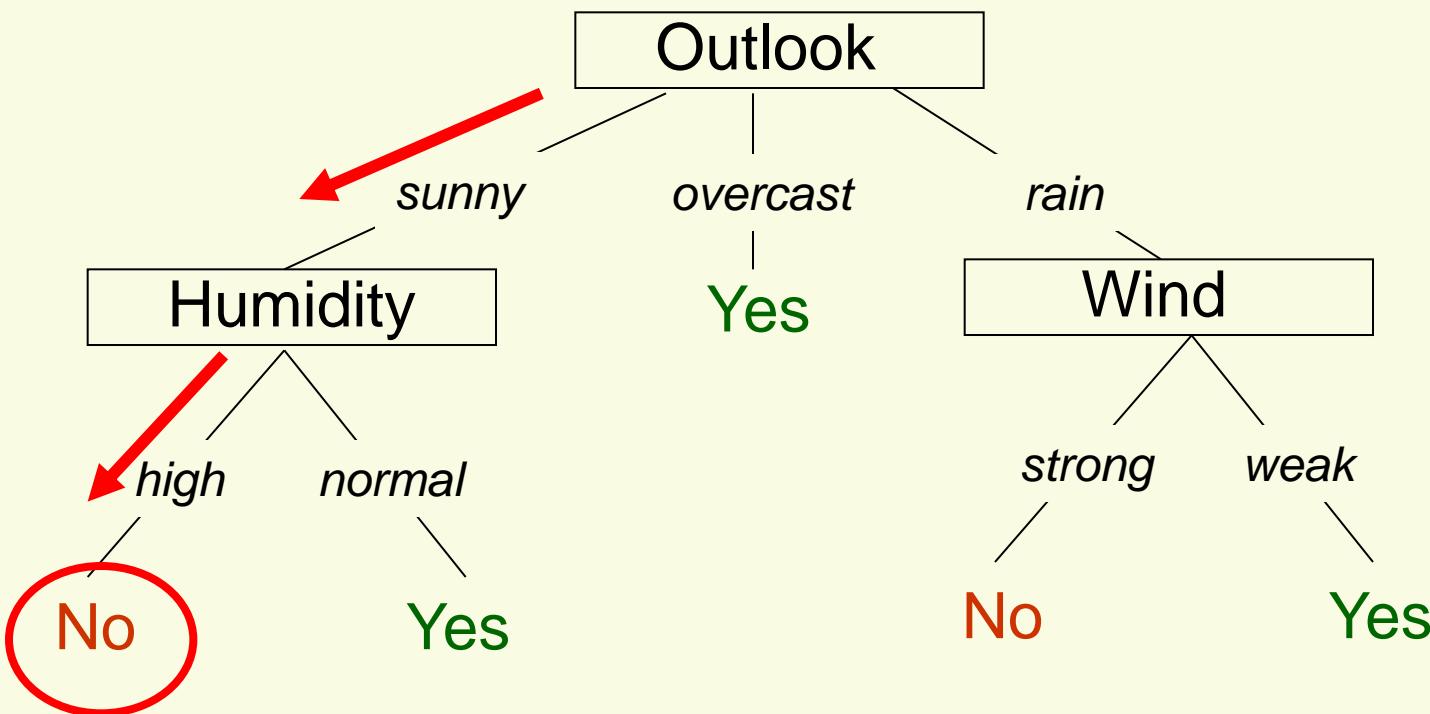
Instance:

(Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong)

Classification :

Play tennis: Yes/No

Decision Tree Representation: Example



Instance:

(Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong)

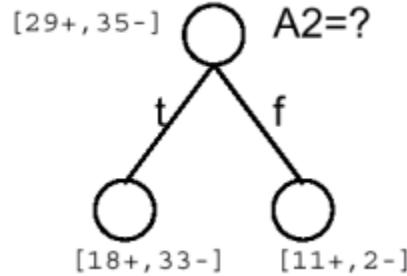
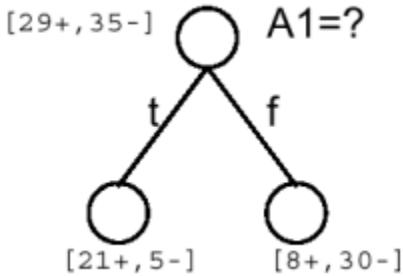
Building a Decision Tree

- I. First test all attributes and select the one that would function as the “best” root;
- II. Main Loop:
 1. $A \leftarrow$ the “best” decision attribute for the next node
 2. Assign A as decision attribute for node
 3. For each value of A, create new descendant of node
 4. Sort training examples to leaf node
 5. Continue this process until the training examples are perfectly classified

Greedy search for an acceptable decision tree

Attribute selection

Which attribute should be taken next, A1 or A2? Which test?



- We seek an attribute that makes the data reaching the immediate descendent nodes as ***pure*** as possible.
- More convenient to define ***impurity*** of a node.
- Let $i(N)$ define the impurity of a node N : in all classes we want $i(N) = 0$ if all the patterns that reach the node belong to the same category, $i(N)$ to be large if all the categories are equally presented.

Entropy Impurity

The most popular measure is *entropy impurity*

$$i(N) = -\sum_{j=1}^n P(w_j) \log_2 P(w_j)$$

Fraction of patterns at node A that are in category w_j



Comes from

Entropy $H(X)$ of a random variable X

$$H(X) = -\sum_{i=1}^n P(X=i) \log_2 P(X=i)$$

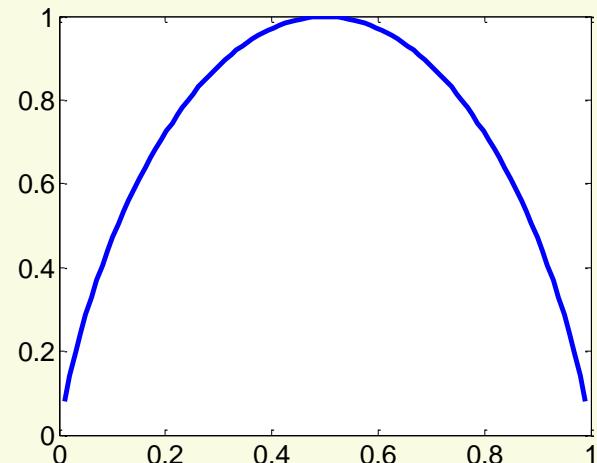
Example of two classes

$$H(X) = -P(w_+) \log P(w_+) - P(w_-) \log P(w_-)$$

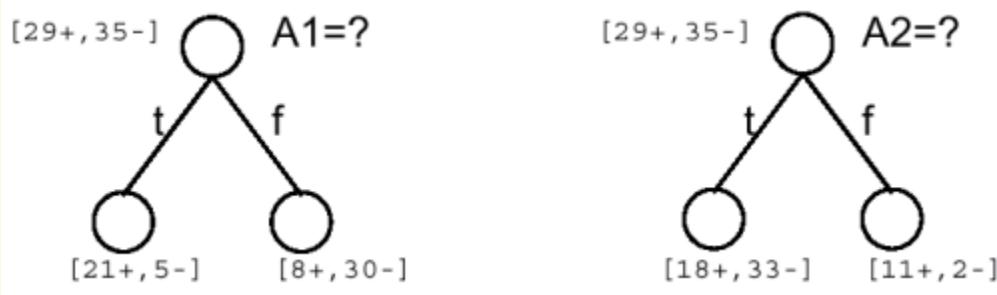
$$P(w_+) = 0, P(w_-) = 1 \rightarrow H(X) = 0$$

$$P(w_+) = 0.5, P(w_-) = 0.5 \rightarrow H(X) = 1$$

Plot of H for $P_+ = 1 - P_-$



Information Gain



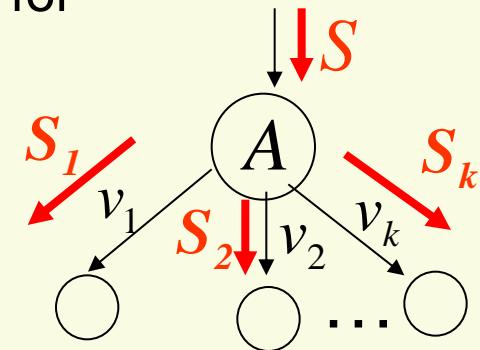
Choose the attribute that most reduces the impurity of the node

Information Gain measured the expected reduction of entropy due to sorting on attribute A

$$Gain(A) = i(N) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} i(N_v)$$

↑ Patterns that reach the node

Subset of S for which $A=v$



Example

Training set:

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Example

- Let $E([X+, Y-])$ represent that there are X positive training elements and Y negative elements.
- Therefore the Entropy for the training data, $E(S)$, can be represented as $E([9+, 5-])$ because of the 14 training examples 9 of them are **yes** and 5 of them are **no**.
- Let's start off by calculating the Entropy of the Training Set.

$$E(S) = E([9+, 5-]) = (-9/14 \log_2 9/14) + (-5/14 \log_2 5/14) = 0.94$$

Example (cont)

- Next we will need to calculate the information gain $G(S,A)$ for each attribute A where A is taken from the set {Outlook, Temperature, Humidity, Wind}.
- The information gain for Outlook is:
 - $$\begin{aligned} G(S, \text{Outlook}) &= E(S) - [5/14 * E(\text{Outlook}=\text{sunny}) + 4/14 * \\ &\quad E(\text{Outlook} = \text{overcast}) + 5/14 * E(\text{Outlook}=\text{rain})] \\ &= E([9+, 5-]) - [5/14 * E(2+, 3-) + 4/14 * E([4+, 0-]) + 5/14 * E([3+, 2-])] \\ &= 0.94 - [5/14 * 0.971 + 4/14 * 0.0 + 5/14 * 0.971] \\ &= \mathbf{0.246} \end{aligned}$$

Example (cont)

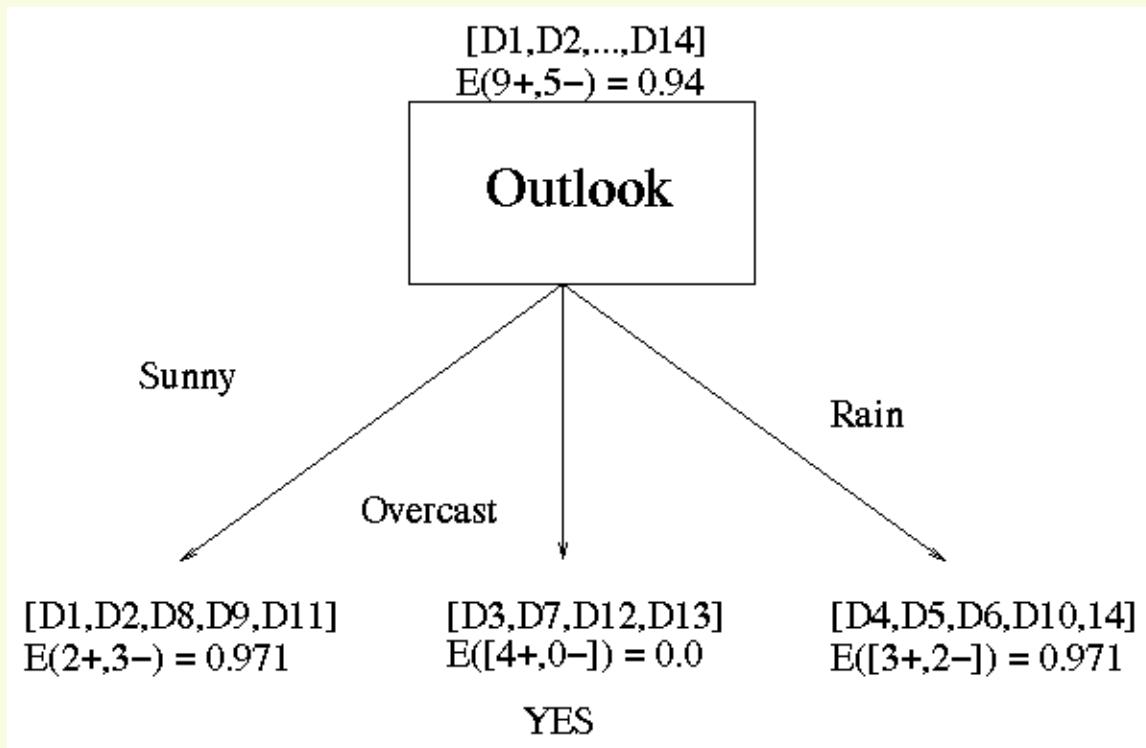
- The information gain for Temperature is:
- $$G(S, \text{Temperature}) = 0.94 - [4/14 * E(\text{Temperature}=\text{hot}) + 6/14 * E(\text{Temperature}=\text{mild}) + 4/14 * E(\text{Temperature}=\text{cool})]$$
- $$\begin{aligned} G(S, \text{Temperature}) &= 0.94 - [4/14 * E([2+, 2-]) + \\ &6/14 * E([4+, 2-]) + 4/14 * E([3+, 1-])] \\ &= 0.94 - [4/14 + 6/14 * 0.918 + 4/14 * 0.811] \\ &= \mathbf{0.029} \end{aligned}$$

Example (cont)

- The information gain for Humidity is:
- $$\begin{aligned} G(S, \text{Humidity}) &= 0.94 - [7/14 * E(\text{Humidity}=\text{high}) + \\ &\quad 7/14 * E(\text{Humidity}=\text{normal})] \\ &= 0.94 - [7/14 * E([3+, 4-]) + 7/14 * E([6+, 1-])] \\ &= 0.94 - [7/14 * 0.985 + 7/14 * 0.592] \\ &= \mathbf{0.1515} \end{aligned}$$
- The information gain for Wind is:
- $$\begin{aligned} G(S, \text{Wind}) &= 0.94 - [8/14 * 0.811 + 6/14 * 1.00] \\ &= \mathbf{0.048} \end{aligned}$$

Example (cont)

- Outlook is our winner!

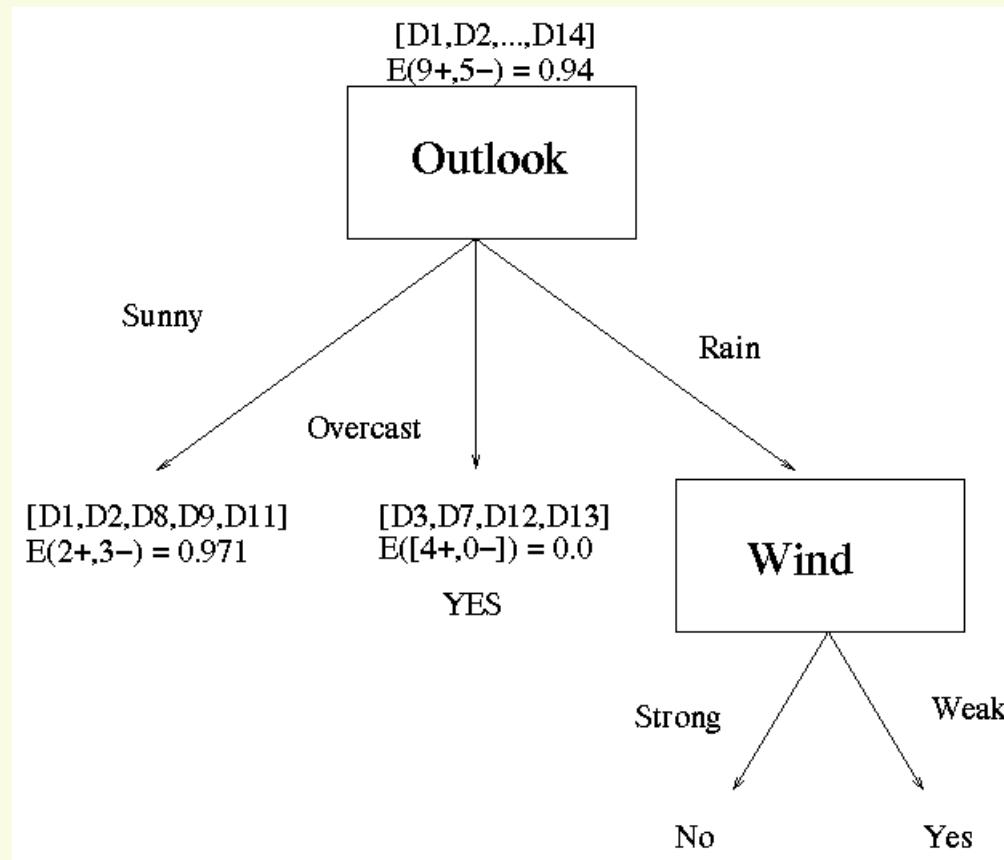


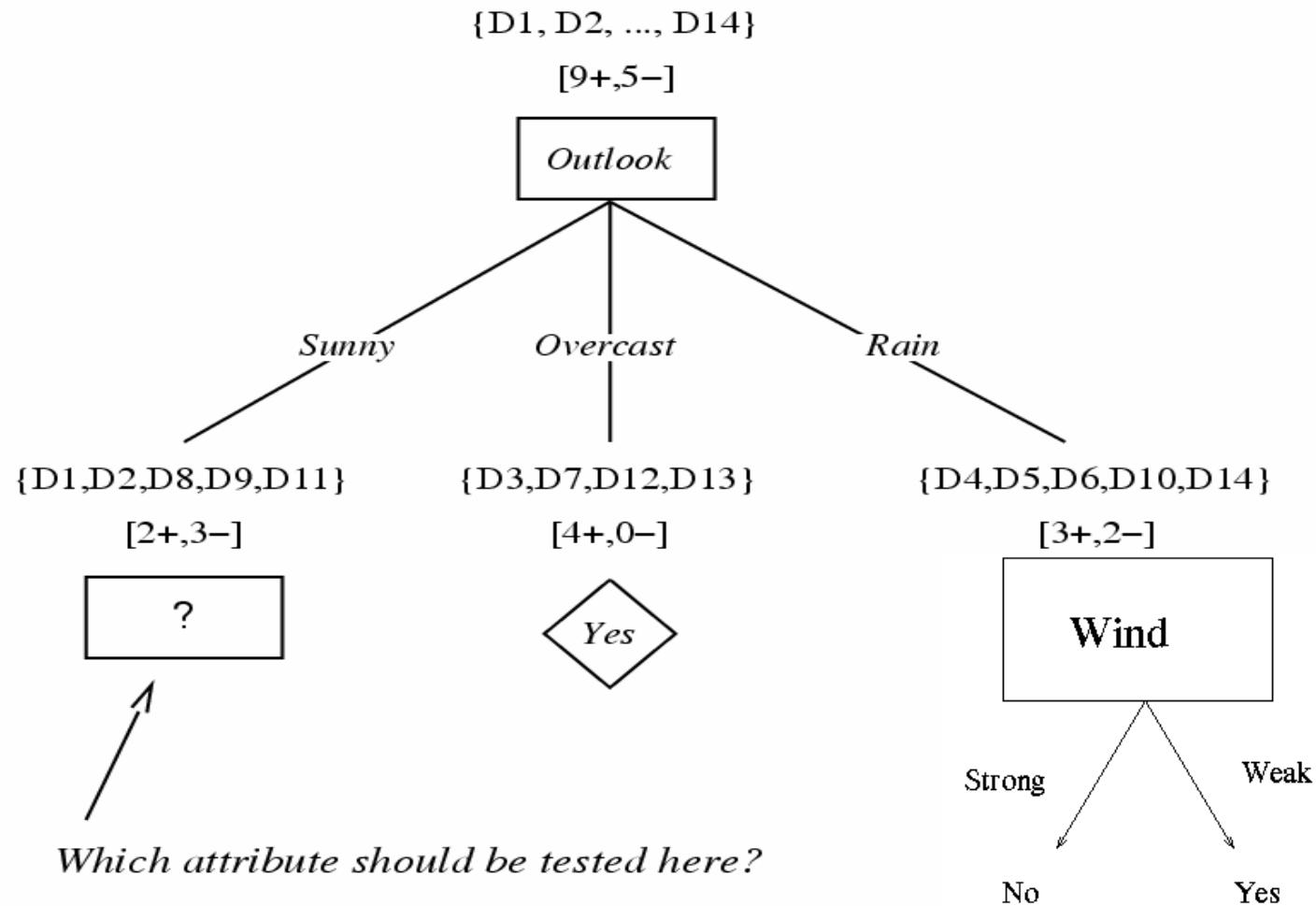
Example (cont)

- Now that we have discovered the root of our decision tree we must now recursively find the nodes that should go below Sunny, Overcast, and Rain.
- $G(\text{Outlook}=\text{Rain}, \text{Humidity}) = 0.971 - [2/5 * E(\text{Outlook}=\text{Rain} \wedge \text{Humidity}=\text{high}) + 3/5 * E(\text{Outlook}=\text{Rain} \wedge \text{Humidity}=\text{normal})] = 0.02$
- $G(\text{Outlook}=\text{Rain}, \text{Wind}) = 0.971 - [3/5 * 0 + 2/5 * 0]$
= 0.971

Example (cont)

- Now our decision tree looks like:



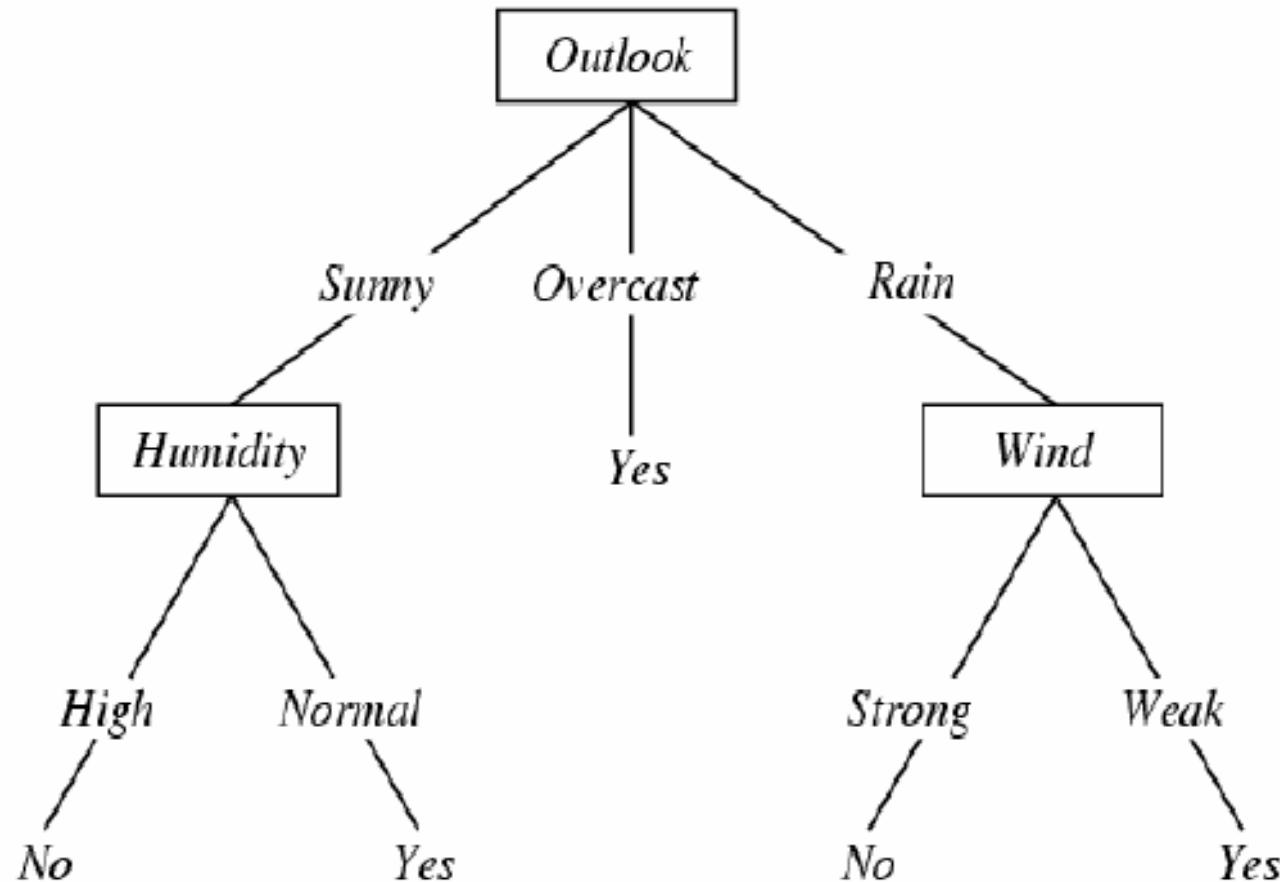


$$S_{sunny} = \{D_1, D_2, D_8, D_9, D_{11}\}$$

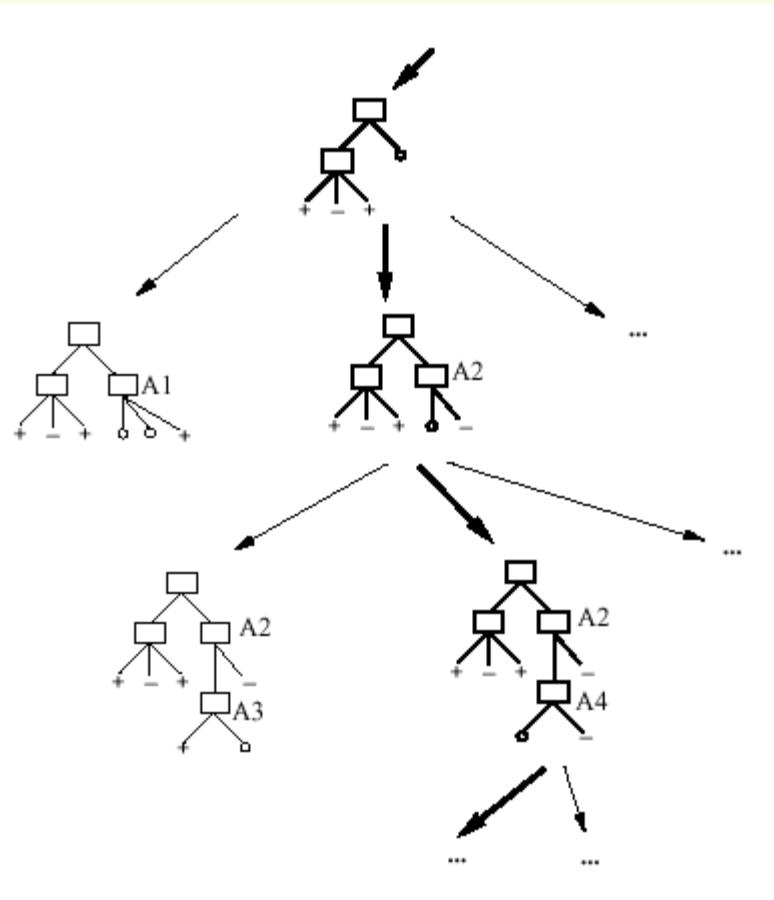
$$Gain(S_{sunny}, \text{Humidity}) = .970 - (3/5) 0.0 - (2/5) 0.0 = .970$$

$$Gain(S_{sunny}, \text{Temperature}) = .970 - (2/5) 0.0 - (2/5) 1.0 - (1/5) 0.0 = .570$$

$$Gain(S_{sunny}, \text{Wind}) = .970 - (2/5) 1.0 - (3/5) .918 = .019$$



Which Tree Should We Output?



- ID3 performs heuristic search through space of decision trees from simplest to increasingly complex, guided by the information gain.
- It stops at smallest acceptable tree.

Occam's razor: prefer the simplest hypothesis that fits the data

Ockham's razor

Why simple trees should be preferred?

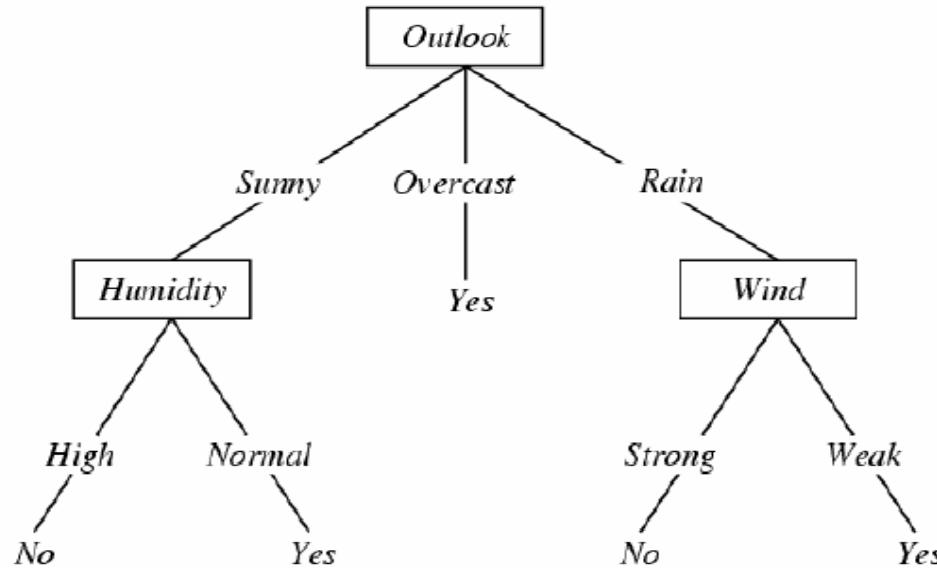
1. The number of simple hypothesis that may accidentally fit the data is small, so chances that simple hypothesis uncover some interesting knowledge about the data are larger.
2. Simpler trees do not partition the feature space into too many small boxes, and thus may generalize better, while complex trees may end up with a separate box for each training data sample.

Overfitting in DT

- Consider adding noisy example

Sunny, Hot, Normal, Strong, PlayTennis=No

- How it effects the earlier tree?



- Noise in data or small number of training examples lead to overfitting

Avoiding overfitting

How to avoid overfitting?

- Stop growing earlier, before it perfectly classifies the training data.
- Grow full tree, then post-prune the tree.

How to select “best” tree:

- Measure performance over separate validation data set.
- Trade complexity for accuracy: minimize $\text{size(tree)} + \text{size(misclassifications(tree))}$
- ...

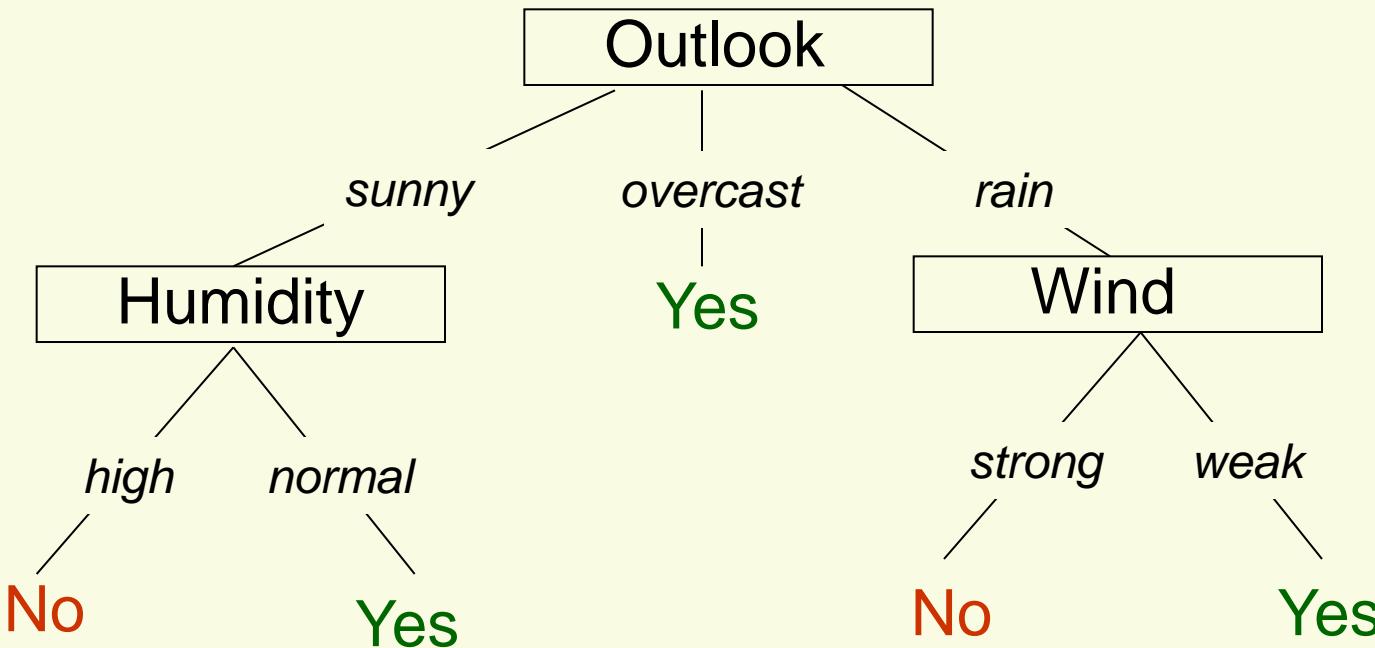
DT Pruning

- One way to deal with overfitting in decision trees is to ***prune*** the tree
- This means to make it smaller by removing nodes that don't turn out to be helpful
- Use a greedy hill-climbing search to prune a tree:
 - for each interior (non-leaf) node n
 - remove the subtree at n
 - replace it with a leaf node marked + or -, according to the majority of the training examples that fall into that subtree.
 - evaluate the performance of the new decision tree on the *validation set*
 - store the pruned tree with the best performance
 - repeat until no improvement is made from removing any interior node
 - if a pruned tree has the same performance as the current tree, then use the pruned tree.

Rule Post-Pruning (used in C4.5)

- Grow the tree until the training data is fit as well as possible.
- Convert the learned tree into an equivalent set of rules, by creating one rule per each path.
- Eliminate unnecessary rule antecedents to simplify the rules.
 - Rules with only one antecedent cannot be further simplified, so we only consider those with two or more.
 - To simplify a rule, eliminate antecedents that have no effect on the conclusion reached by the rule.
- Sort final rules into desired sequence for use.

Converting Tree to Rules



IF $(Outlook=Sunny) \text{ AND } (Humidity=High)$

THEN $PlayTennis = No$

IF $(Outlook=Sunny) \text{ AND } (Humidity=Normal)$

THEN $PlayTennis = Yes$

Boosting

Some slides are due to Robin Dhamankar
Vandi Verma & Sebastian Thrun

Boosting: motivation

- It is usually hard to design an accurate classifier which generalizes well
- However it is usually easy to find many “rule of thumb” **weak** classifiers
 - A classifier is **weak** if it is only slightly better than random guessing
- Can we combine several weak classifiers to produce an accurate classifier?
 - Question people have been working on since 1980's

Ada Boost

- Let's assume we have 2-class classification problem, with $y_i \in \{-1, 1\}$
- Ada boost will produce a discriminant function:

$$g(x) = \sum_{t=1}^T \alpha_t f_t(x), \quad \alpha_t \geq 0$$

where $f_t(x)$ is the “weak” classifier

- The final classifier is sign of $g(x)$
- Given x , each weak classifier votes for a label $f_t(x)$ using α_t votes allocated to it. The ensemble then classifies the example according to which label receives the most votes.
- Note that $g(x) \in [-1, 1]$ whenever the votes are normalized to sum to one. So, $g(x) = 1$ only if all the weak classifiers agree that the label should be $y = 1$.

Idea Behind Ada Boost

- Algorithm is iterative
- Maintains distribution of weights over the training examples
- Initially distribution of weights is uniform
- At successive iterations, the weight of misclassified examples is increased, forcing the weak learner to focus on the hard examples in the training set

More Comments on Ada Boost

- Ada boost is very simple to implement, provided you have an implementation of a “weak learner”
- Will work as long as the “basic” classifier $f_t(x)$ is at least slightly better than random
- Can be applied to boost any classifier, not necessarily weak

Ada Boost (*slightly modified from the original version*)

- $d(x)$ is the distribution of weights over the N training points $\sum d(x_i) = 1$
- Initially assign uniform weights $d_0(x_i) = 1/N$ for all x_i
- At each iteration t :
 - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
 - Compute the error rate ε_t as
$$\varepsilon_t = \sum_{i=1 \dots N} d_t(x_i) \cdot I[y_i \neq f_t(x_i)]$$
 - assign weight α_t to the classifier f_t 's in the final hypothesis
$$\alpha_t = \frac{1}{2} \log ((1 - \varepsilon_t)/\varepsilon_t)$$
 - For each x_i , $d_{t+1}(x_i) = d_t(x_i) \cdot \exp(-\alpha_t y_i f_t(x_i))$
 - Normalize $d_{t+1}(x_i)$ so that $\sum_{i=1} d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign} [\sum \alpha_t f_t(x)]$

Ada Boost

- At each iteration t :
 - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
 - Compute ε_t the error rate as
$$\varepsilon_t = \sum d_t(x_i) \cdot I[y_i \neq f_t(x_i)]$$
 - assign weight α_t the classifier f_t 's in the final hypothesis
$$\alpha_t = \frac{1}{2} \log ((1 - \varepsilon_t)/\varepsilon_t)$$
 - For each x_i , $d_{t+1}(x_i) = d_t(x_i) \cdot \exp(-\alpha_t y_i f_t(x_i))$
 - Normalize $d_{t+1}(x_i)$ so that $\sum_{t+1} d(x_i) = 1$
- $f_{FINAL}(x) = \text{sign} [\sum \alpha_t f_t(x)]$

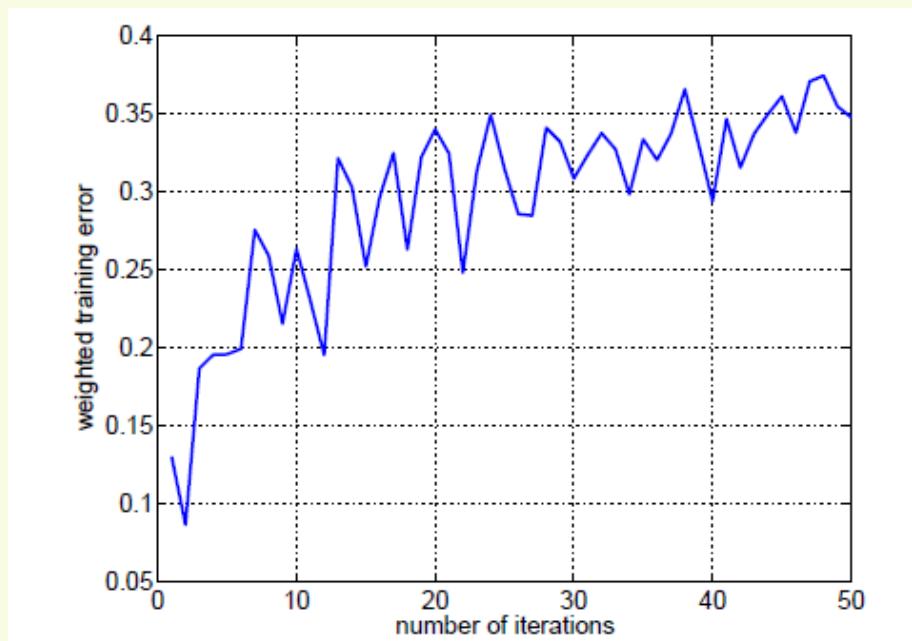
- If the classifier does not take weighted samples, this step can be achieved by sampling from the training samples according to the distribution $d_t(x)$

Ada Boost

- At each iteration t :
 - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
 - Compute ε_t the error rate as
$$\varepsilon_t = \sum d_t(x_i) \cdot I[y_i \neq f_t(x_i)]$$
 - assign weight α_t the classifier f_t 's in the final hypothesis
$$\alpha_t = \frac{1}{2} \log ((1 - \varepsilon_t)/\varepsilon_t)$$
 - For each x_i , $d_{t+1}(x_i) = d_t(x_i) \cdot \exp(-\alpha_t y_i f_t(x_i))$
 - Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign} [\sum \alpha_t f_t(x)]$
- Since the weak classifier is better than random, we expect $\varepsilon_t < 1/2$

Weighted error (ε_t)

- The weighted error achieved by a new simple classifier $f_t(x)$ relative to weights $d_t(x)$ tends to increase with t , i.e., with each boosting iteration (though not monotonically).
- The reason for this is that since the weights concentrate on examples that are difficult to classify correctly, subsequent base learners face harder classification tasks.



Weighted error (ε_t)

- It can be shown that the weighted error of the simple classifier $f_t(x)$ relative to updated weights $d_{t+1}(x)$ is exactly 0.5.
- This means that the simple classifier introduced at the t -th boosting iteration will be useless (at chance level) for the next boosting iteration. So the boosting algorithm would never introduce the same simple classifier twice in a row.
- It could, however, reappear later on (relative to a different set of weights)

Ada Boost

- At each iteration t :

- Find best weak classifier $f_t(x)$ using weights $d_t(x)$

- Compute ε_t the error rate as

$$\varepsilon_t = \sum d(x_i) \cdot I(y_i \neq f_t(x_i))$$

- assign weight α_t the classifier f_t 's in the final hypothesis

$$\alpha_t = \frac{1}{2} \log \left(\frac{(1 - \varepsilon_t)}{\varepsilon_t} \right)$$

- For each x_i , $d_{t+1}(x_i) = d_t(x_i) \cdot \exp(-\alpha_t y_i f_t(x_i))$

- Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$

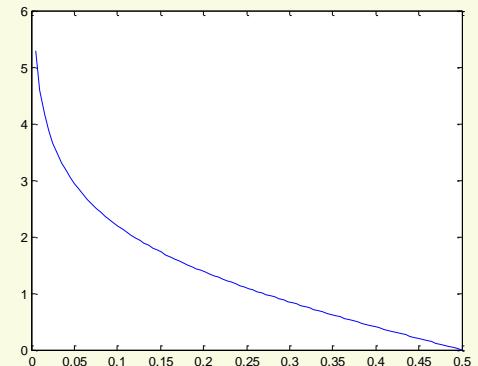
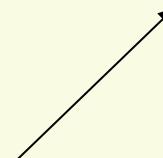
- $f_{FINAL}(x) = \text{sign} [\sum \alpha_t f_t(x)]$

- Recall that $\varepsilon_t < \frac{1}{2}$

- Thus $(1 - \varepsilon_t)/\varepsilon_t > 1 \Rightarrow \alpha_t > 0$

- The smaller is ε_t , the larger is α_t , and thus the more importance (weight) classifier $f_t(x)$ gets in the final classifier

$$f_{FINAL}(x) = \text{sign} [\sum \alpha_t f_t(x)]$$



Ada Boost

- At each iteration t :
 - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
 - Compute ε_t the error rate as
$$\varepsilon_t = \sum d_t(x_i) \cdot I(y_i \neq f_t(x_i))$$
 - assign weight α_t the classifier f_t 's in the final hypothesis
$$\alpha_t = \frac{1}{2} \log ((1 - \varepsilon_t)/\varepsilon_t)$$
 - For each x_i , $d_{t+1}(x_i) = d_t(x_i) \cdot \exp(-\alpha_t y_i f_t(x_i))$
 - Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign} [\sum \alpha_t f_t(x)]$
- Weight of misclassified examples is increased and the new $d_{t+1}(x_i)$'s are normalized to be a distribution again

Ada Boost

- At each iteration t :
 - Find best weak classifier $f_t(x)$ using weights $d_t(x)$
 - Compute ε_t the error rate as
$$\varepsilon_t = \sum d_t(x_i) \cdot I(y_i \neq f_t(x_i))$$
 - assign weight α_t the classifier f_t 's in the final hypothesis
$$\alpha_t = \frac{1}{2} \log ((1 - \varepsilon_t)/\varepsilon_t)$$
 - For each x_i , $d_{t+1}(x_i) = d_t(x_i) \cdot \exp(-\alpha_t y_i f_t(x_i))$
 - Normalize $d_{t+1}(x_i)$ so that $\sum d_{t+1}(x_i) = 1$
- $f_{FINAL}(x) = \text{sign} [\sum \alpha_t f_t(x)]$

Ensemble training error

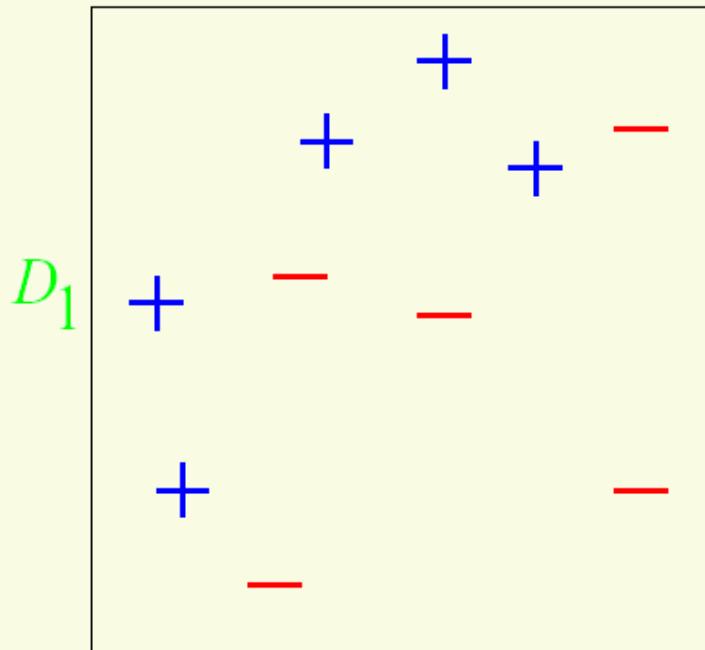
- It can be shown that the training error drops exponentially fast, if each weak classifier is slightly better than random

$$Err_{train} \leq \exp\left(-2\sum_t \gamma_t^2\right)$$

- Here $\gamma_t = \varepsilon_t - 1/2$, where ε_t is classification error at round t (weak classifier f_t)

AdaBoost Example

from “A Tutorial on Boosting” by Yoav Freund and Rob Schapire

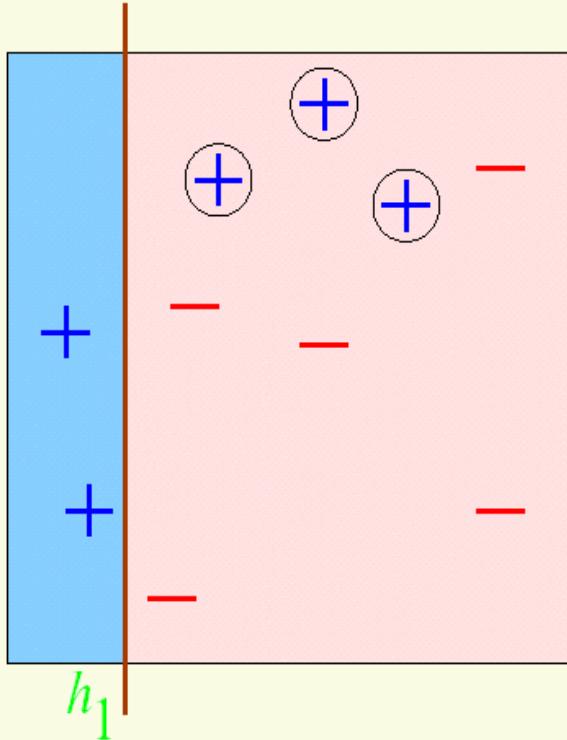


Original Training set : equal weights to all training samples

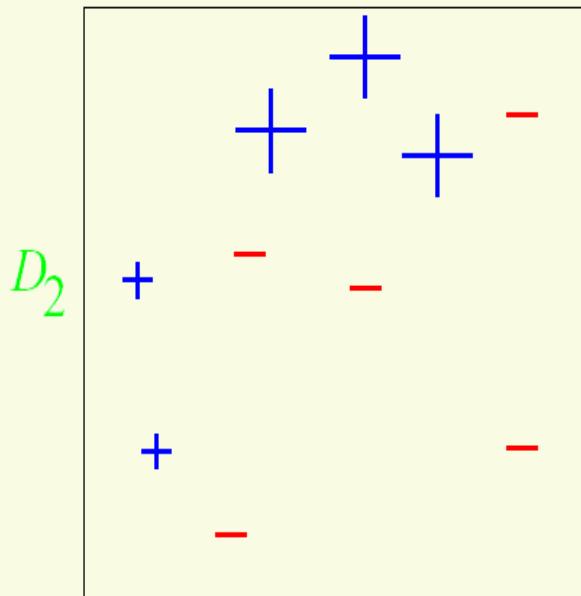
Note: in the following slides, $h_t(x)$ is used instead of $f_t(x)$, and D instead of d

AdaBoost Example

ROUND 1

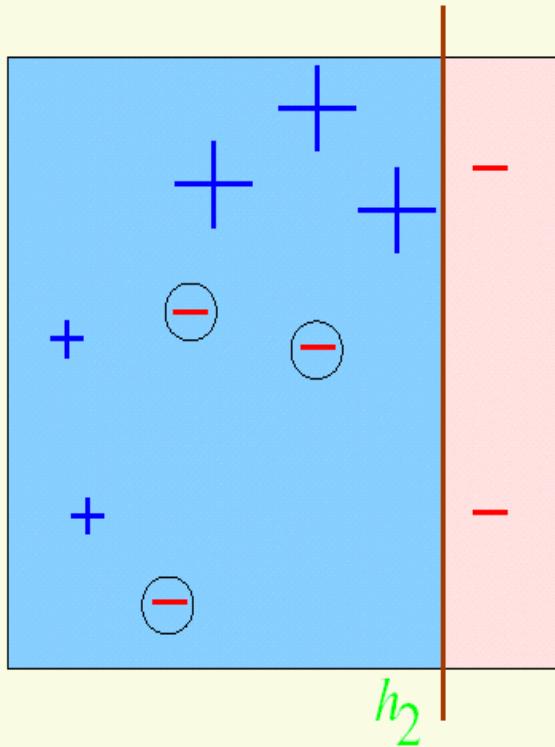


$$\begin{aligned}\varepsilon_1 &= 0.30 \\ \alpha_1 &= 0.42\end{aligned}$$

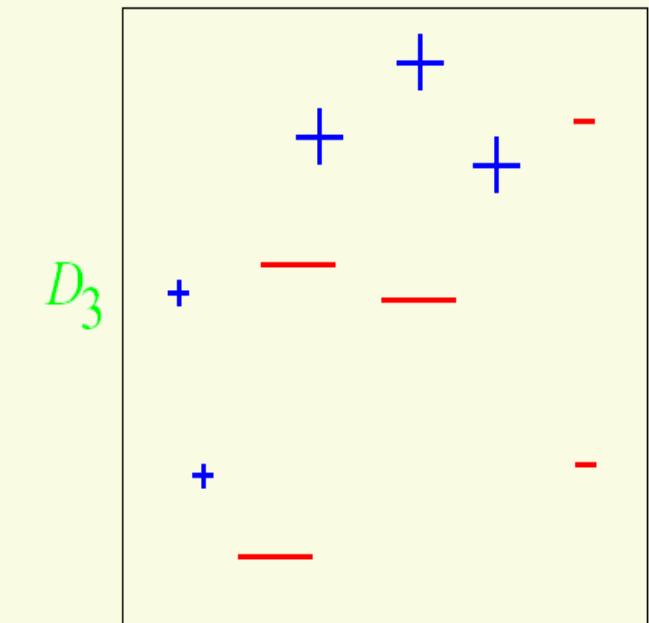


AdaBoost Example

ROUND 2

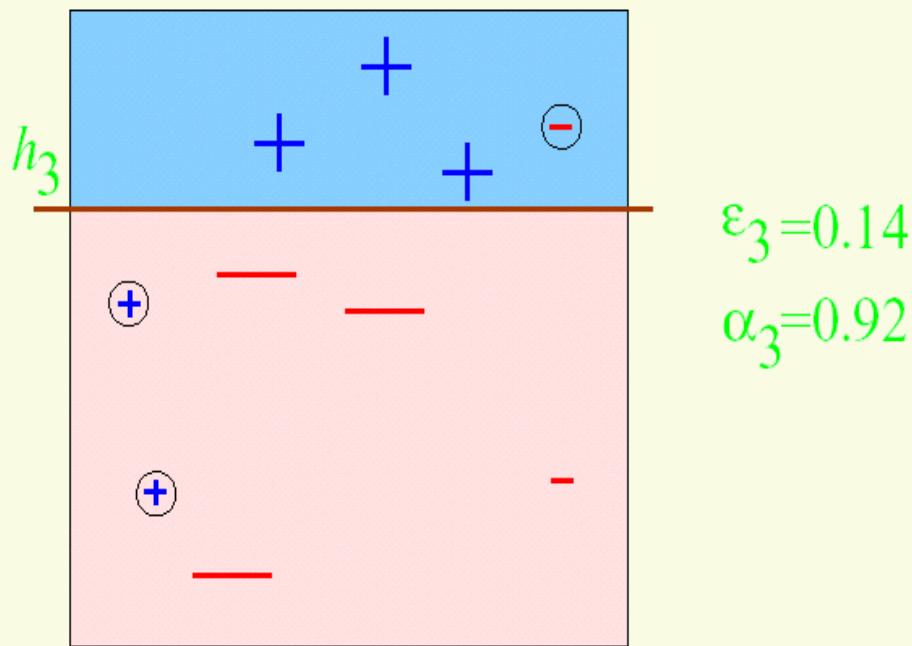


$$\begin{aligned}\varepsilon_2 &= 0.21 \\ \alpha_2 &= 0.65\end{aligned}$$

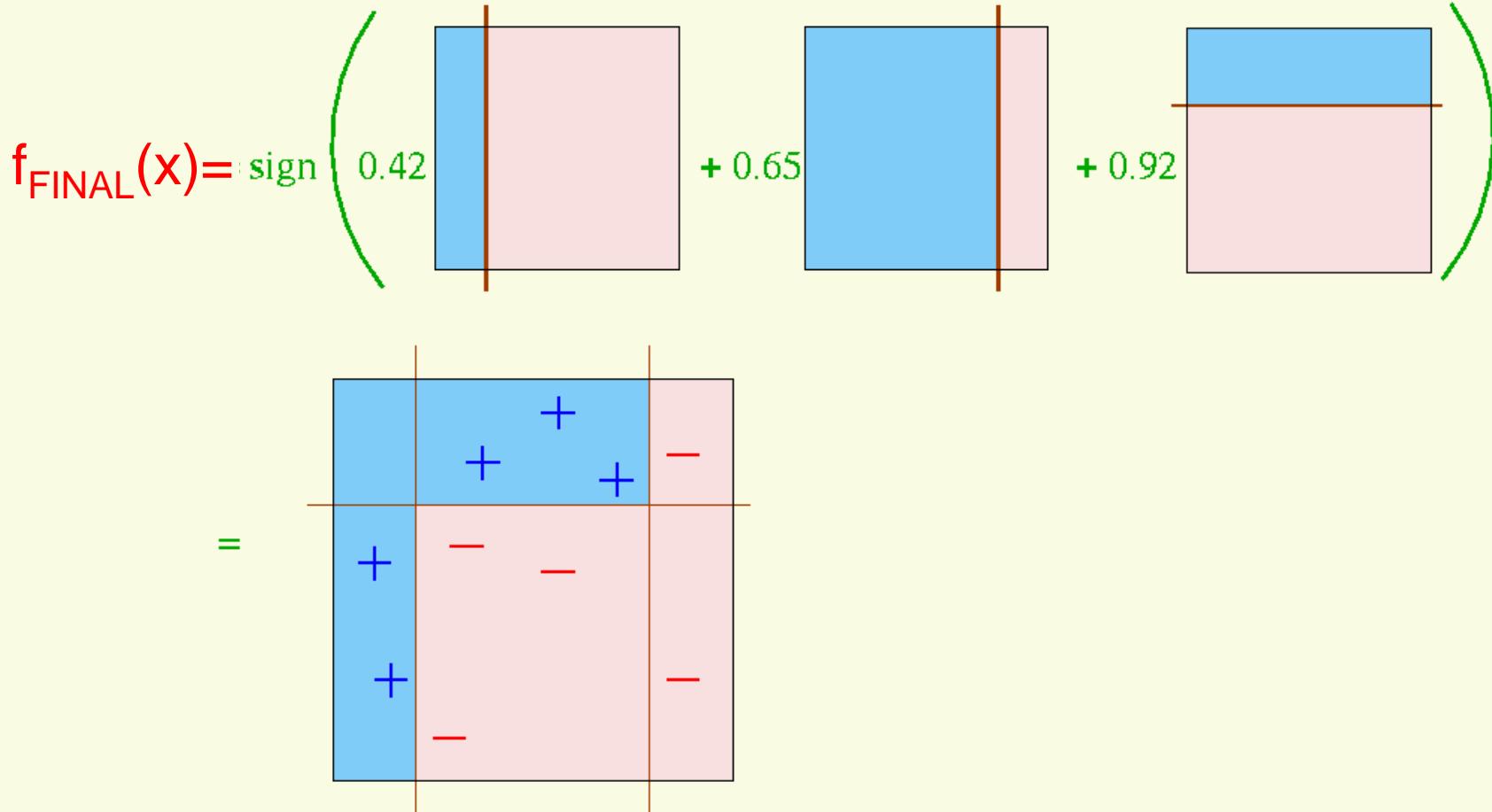


AdaBoost Example

ROUND 3



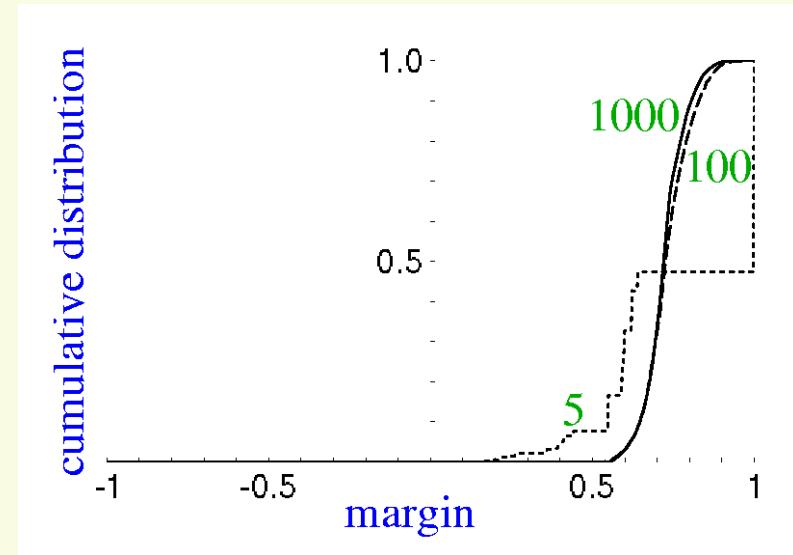
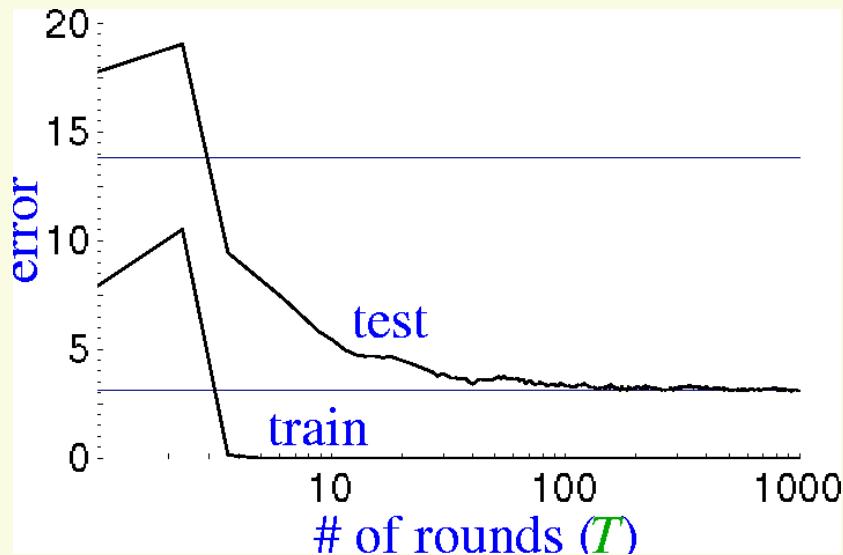
AdaBoost Example



AdaBoost Comments

- But we are really interested in the generalization properties of $f_{\text{FINAL}}(x)$, not the training error
- AdaBoost was shown to have excellent generalization properties in practice.
- It can be shown that boosting “aggressively” increases the margins of training examples, as iterations proceed
 - margins continue to increase even when training error reaches zero
 - Helps to explain empirically observed phenomena: test error continues to drop even after training error reaches zero

The Margin Distribution



epoch	5	100	1000
training error	0.0	0.0	0.0
test error	8.4	3.3	3.1
%margins ≤ 0.5	7.7	0.0	0.0
Minimum margin	0.14	0.52	0.55

Practical Advantages of AdaBoost

- fast
- simple
- Has only one parameter to tune (T)
- flexible: can be combined with any classifier
- provably effective (assuming weak learner)
 - shift in mind set: goal now is merely to find hypotheses that are better than random guessing
- finds outliers
 - The hardest examples are frequently the “outliers”

Caveats

- performance depends on data & weak learner
- AdaBoost can fail if
 - weak hypothesis too complex (overfitting)
 - weak hypothesis too weak
- empirically, AdaBoost seems especially susceptible to noise

Linear Regression

Linear Regression with Shrinkage

Some slides are due to Tommi Jaakkola, MIT AI Lab

Introduction

- The goal of **regression** is to make quantitative (real valued) predictions on the basis of a (vector of) features or attributes.
- **Examples:** house prices, stock values, survival time, fuel efficiency of cars, etc.

Predicting vehicle fuel efficiency (mpg) from 8 attributes:

y	x				
	cyls	disp	hp	weight	...
18.0	8	307.0	130.00	3504	...
26.0	4	97.00	46.00	1835	...
33.5	4	98.00	83.00	2075	...
...					

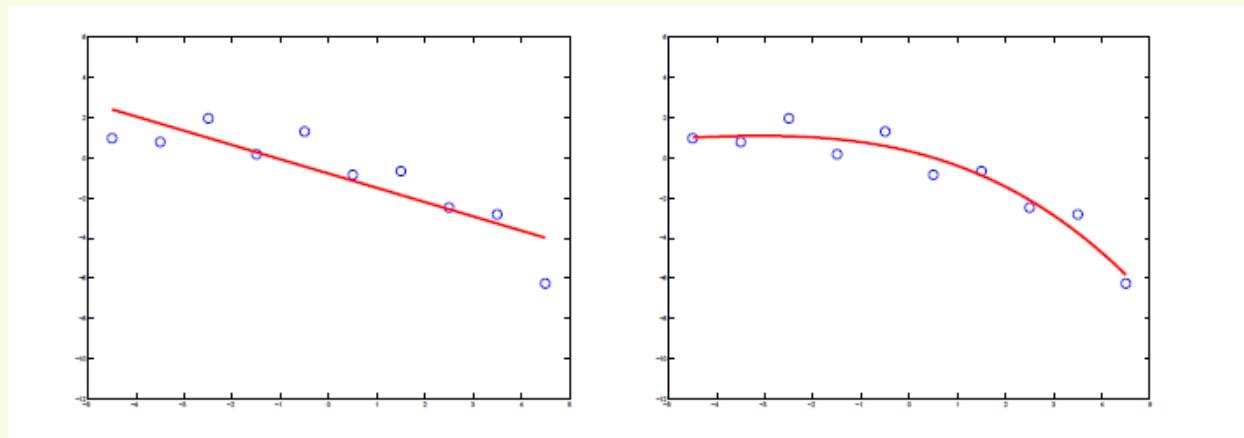
A generic regression problem

- The input attributes are given as fixed length vectors $x \in \Re^M$ that could come from different sources: inputs, transformation of inputs (log, square root, etc...) or basis functions.
- The outputs are assumed to be real valued $y \in \Re$ (with some possible restrictions).
- Given n iid training samples $D = \{(x_1, y_1) \dots (x_n, y_n)\}$ from unknown distribution $P(x, y)$, the goal is to minimize the prediction error (loss) on new examples (x, y) drawn at random from the same $P(x, y)$.
- An example of a loss function:

$$L(f(x), y) = (f(x) - y)^2 \quad \text{squared loss}$$

↑
our prediction for x

Regression Function



- We need to define a class of functions (types of predictions we make).

Linear prediction:

$$f(x; w_0, w_1) = w_0 + w_1 x$$

where w_0, w_1 are the parameters we need estimate.

Linear Regression

Typically we have a set of training data $(x_1, y_1) \dots (x_n, y_n)$ from which we estimate the parameters w_0, w_1 .

Each $x_i = (x_{i1}, \dots, x_{iM})$ is a vector of measurements for i th case.

or

$h(x_i) = \{h_0(x_i), \dots, h_M(x_i)\}$ a basis expansion of x_i

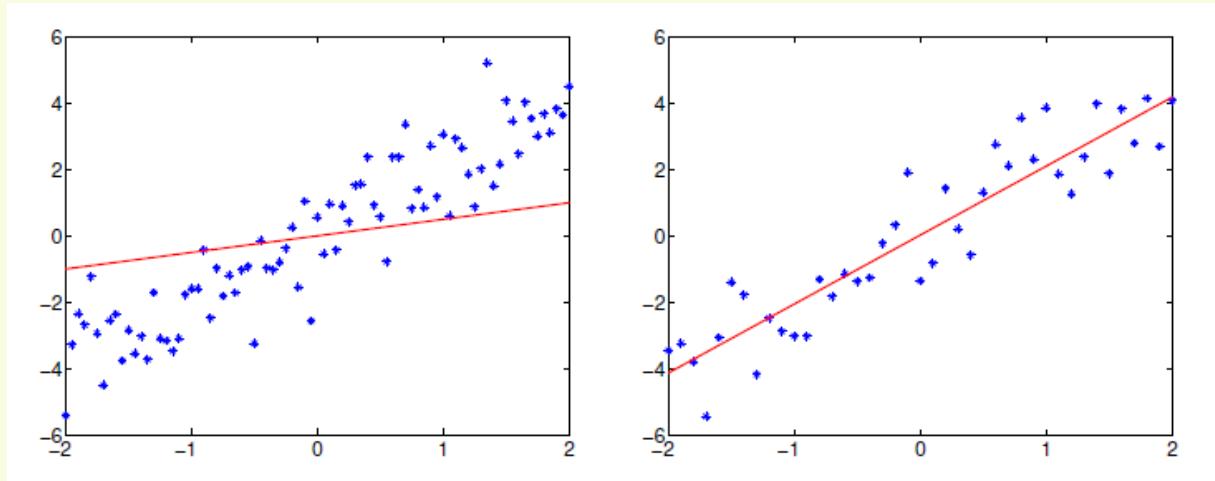
$$y(x) \approx f(x; w) = w_0 + \sum_{j=1}^M w_j h_j(x) = w^t h(x)$$

(define $h_0(x) = 1$)

Basis Functions

- There are many basis functions we can use e.g.
 - Polynomial $h_j(x) = x^{j-1}$
 - Radial basis functions $h_j(x) = \exp\left(-\frac{(x - \mu_j)^2}{2s^2}\right)$
 - Sigmoidal $h_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$
 - Splines, Fourier, Wavelets, etc

Estimation Criterion



- We need a fitting/estimation criterion to select appropriate values for the parameters w_0, w_1 based on the training set $D = \{(x_1, y_1) \dots (x_n, y_n)\}$
- For example, we can use the empirical loss:

$$J_n(w_1, w_0) = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i; w_1, w_0))^2$$

(note: the loss is the same as in evaluation)

Empirical loss: motivation

- Ideally, we would like to find the parameters w_0, w_1 , that minimize the expected loss (unlimited training data):

$$J(w_0, w_1) = E_{(x,y) \sim P} (y_i - f(x_i; w_0, w_1))^2$$

where the expectation is over samples from $P(x,y)$.

- When the number of training examples n is large:

$$E_{(x,y) \sim P} (y_i - f(x_i; w_0, w_1))^2 \approx \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i; w_0, w_1))^2$$

Expected loss

Empirical loss

Linear Regression Estimation

- Minimize the empirical squared loss

$$\begin{aligned} J_n(w_0, w_1) &= \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i; w_0, w_1))^2 \\ &= \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2 \end{aligned}$$

- By setting the derivatives with respect to w_1, w_0 to zero we get necessary conditions for the “optimal” parameter values.

$$\frac{\partial}{\partial w_1} J_n(w_0, w_1) = \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)(-x_i) = 0$$

$$\frac{\partial}{\partial w_0} J_n(w_0, w_1) = \frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)(-1) = 0$$

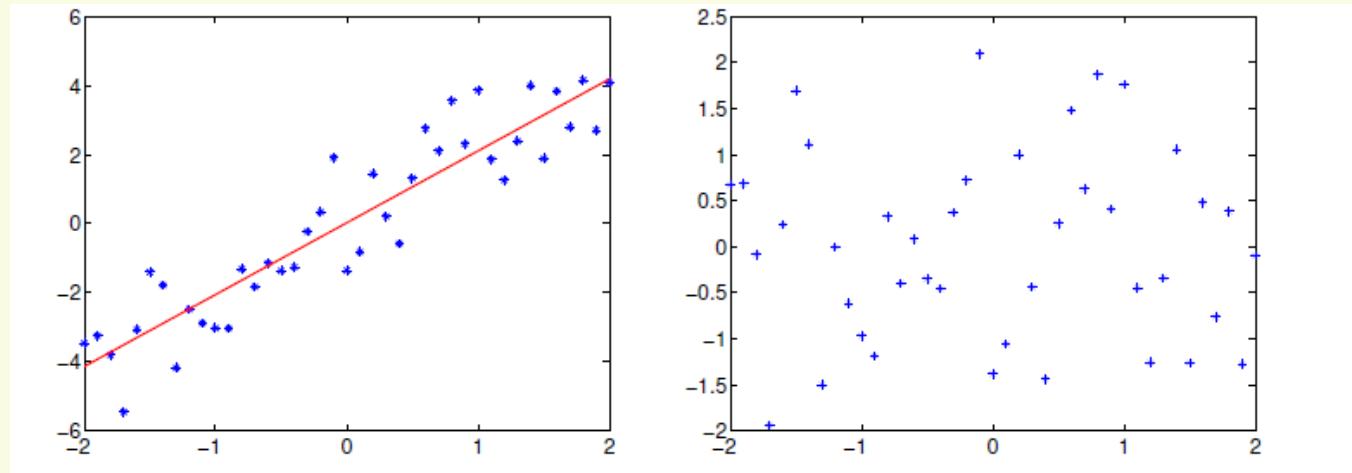
Interpretation

The optimality conditions

$$\frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)(-x_i) = 0$$

$$\frac{2}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)(-1) = 0$$

ensure that the prediction error $\varepsilon_i = (y - w_0 - w_1 x_i)$ is decorrelated with any linear function of the inputs.



Linear Regression: Matrix Form

$$X = \begin{bmatrix} 1 & -x_1 - \\ \dots & \dots \\ 1 & -x_n - \\ & \vdots & \end{bmatrix}_{M+1}^{n \text{ samples}}$$
$$W = \begin{bmatrix} w_0 \\ | \\ w_1 \\ | \end{bmatrix}_{M+1}$$
$$y = \begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix}$$

$$J_n(w_0, w_1) = \frac{1}{n} \sum_{i=1}^n (y_i - w_0 - w_1 x_i)^2 = (Xw - y)^t (Xw - y)$$

Linear Regression Solution

- By setting the derivatives of $(Xw - y)^t(Xw - y)$ to zero,

$$\frac{2}{n}(X^t y - X^t X w) = 0$$

we get the solution:

$$\hat{w} = (X^t X)^{-1} X^t y$$

The solution is a linear function of the outputs y .

Statistical view of linear regression

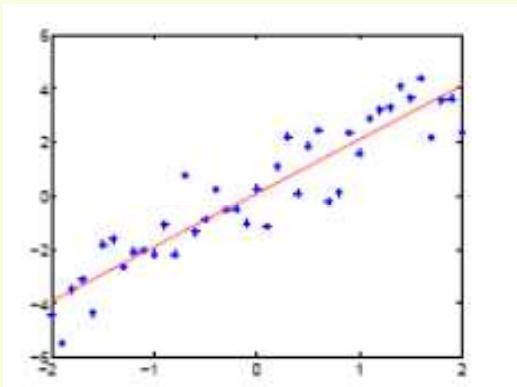
- In a statistical regression model we model both the function and noise

Observed output = function + noise

$$y(x) = f(x; w) + \varepsilon$$

where, e.g., $\varepsilon \sim N(0, \sigma^2)$

- Whatever we cannot capture with our chosen family of functions will be interpreted as noise

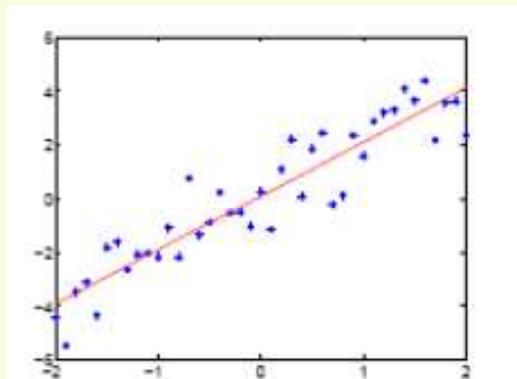


Statistical view of linear regression

- $f(x; w)$ is trying to capture the mean of the observations y given the input x :

$$E[y | x] = E[f(x; w) + \varepsilon | x] = f(x; w)$$

- where $E[y | x]$ is the conditional expectation of y given x , evaluated according to the model (not according to the underlying distribution of X)



Statistical view of linear regression

- According to our statistical model

$$y(x) = f(x; w) + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2)$$

the outputs y given x are normally distributed with mean $f(x; w)$ and variance σ^2 :

$$p(y | x, w, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(y - f(x; w))^2\right]$$

(we model the uncertainty in the predictions, not just the mean)

Maximum likelihood estimation

- Given observations $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ we find the parameters w that maximize the likelihood of the outputs:

$$\begin{aligned} L(w, \sigma^2) &= \prod_{i=1}^n p(y_i | x_i, w, \sigma^2) \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \exp \left\{ -\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - f(x_i; w))^2 \right\} \end{aligned}$$

- Maximize log-likelihood

$$\log L(w, \sigma^2) = \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n - \left(\frac{1}{2\sigma^2} \sum_{i=1}^n (y_i - f(x_i; w))^2 \right)$$

minimize

Maximum likelihood estimation

- Thus

$$w_{MLE} = \arg \min_w \sum_{i=1}^n (y_i - f(x_i; w))^2$$

- But the empirical squared loss is

$$J_n(w) = \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i; w))^2$$

Least-squares Linear Regression is MLE for Gaussian noise !!!

Linear Regression (is it “good”?)

- Simple model
- Straightforward solution

BUT

- MLS is not a good estimator for prediction error
- The matrix $X^T X$ could be ill conditioned
 - Inputs are correlated
 - Input dimension is large
 - Training set is small

Linear Regression - Example

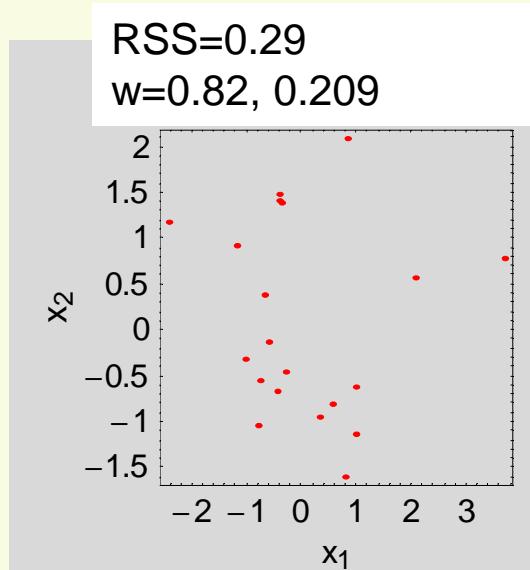
In this example the output is generated by the model (where ε is a small white Gaussian noise):

$$y = 0.8x_1 + 0.2x_2 + \varepsilon$$

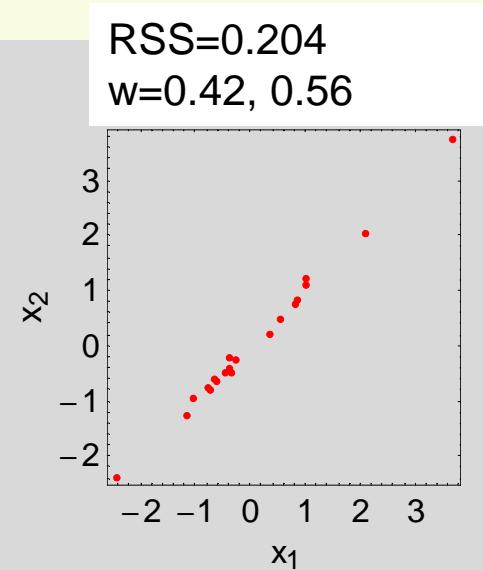
Three training sets with different correlations between the two inputs were randomly chosen, and the linear regression solution was applied.

Linear Regression – Example

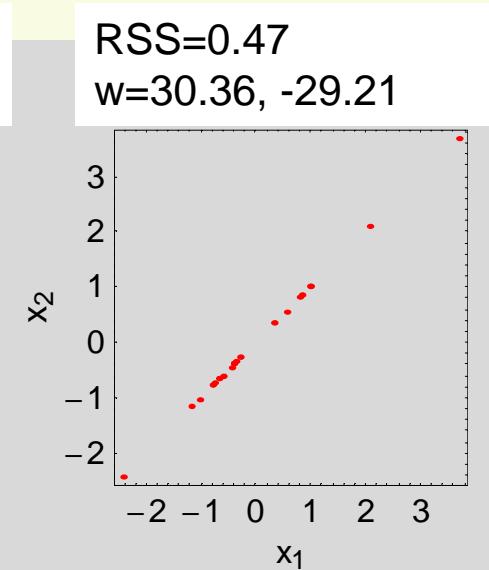
And the results are...



x_1, x_2 uncorrelated



x_1, x_2 correlated



x_1, x_2 strongly correlated

Strong correlation can cause the coefficients to be very large, and they will cancel each other to achieve a good RSS.

Linear Regression

What can be done?

Shrinkage methods

- Ridge Regression
- Lasso
- PCR (Principal Components Regression)
- PLS (Partial Least Squares)

Shrinkage methods

Before we proceed:

- Since the following methods are not invariant under input scale, we will assume the input is normalized (mean 0, variance 1):

$$x'_{ij} \leftarrow x_{ij} - \bar{x}_j \quad x_{ij} \leftarrow \frac{x'_{ij}}{\sigma_j}$$

- The offset w_0 is always estimated as $w_0 = (\sum_i y_i) / N$ and we will work with centered y (meaning $y^i \leftarrow y - w_0$)

Ridge Regression

- Ridge regression shrinks the regression coefficients by imposing a penalty on their size (also called weight decay)
- In ridge regression, we add a quadratic penalty on the weights:

$$J(w) = \sum_{i=1}^N \left(y_i - w_0 - \sum_{j=1}^M x_{ij} w_j \right)^2 + \lambda \sum_{j=1}^M w_j^2$$

where $\lambda \geq 0$ is a tuning parameter that controls the amount of shrinkage.

- The size constraint prevents the phenomenon of wildly large coefficients that cancel each other from occurring.

Ridge Regression Solution

- Ridge regression in matrix form:

$$J(w) = (y - Xw)^t (y - Xw) + \lambda w^t w$$

- The solution is

$$\hat{w}^{ridge} = (X^t X + \lambda I_M)^{-1} X^t y$$

$$\hat{w}^{LS} = (X^t X)^{-1} X^t y$$

- The solution adds a positive constant to the diagonal of $X^T X$ before inversion. This makes the problem non singular even if X does not have full column rank.
- For orthogonal inputs the ridge estimates are the scaled version of least squares estimates:

$$\hat{w}^{ridge} = \gamma \hat{w}^{LS} \quad 0 \leq \gamma \leq 1$$

Ridge Regression (insights)

The matrix X can be represented by it's SVD:

$$X = UDV^T$$

- U is a $N*M$ matrix, it's columns span the column space of X
- D is a $M*M$ diagonal matrix of singular values
- V is a $M*M$ matrix, it's columns span the row space of X

Lets see how this method looks in the Principal Components coordinates of X

Ridge Regression (insights)

$$X' = XV$$

$$Xw = (XV)(V^T w) = X'w'$$

The least squares solution is given by:

$$\hat{w}^{ls} = V^T \hat{w}^{ls} = V^T (VDU^T UDV^T)^{-1} VDU^T y = D^{-1} U^T y$$

The Ridge Regression is similarly given by:

$$\begin{aligned}\hat{w}^{ridge} &= V^T \hat{w}^{ridge} = V^T (VDU^T UDV^T + \lambda I)^{-1} VDU^T y = \\ &= (D^2 + \lambda I)^{-1} DU^T y = \boxed{(D^2 + \lambda I)^{-1} D^2 \hat{w}^{ls}}\end{aligned}$$

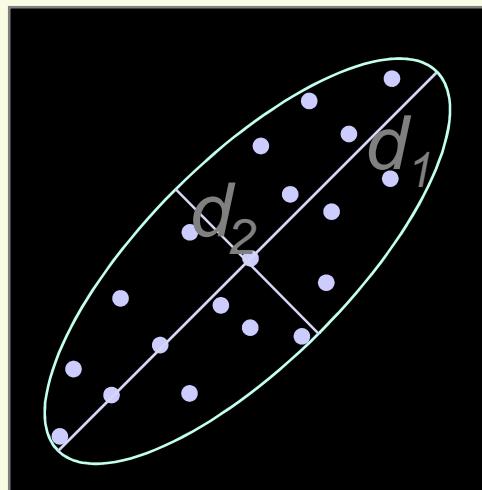
Diagonal

Ridge Regression (*insights*)

$$\hat{w}_j^{ridge} = \frac{d_j^2}{d_j^2 + \lambda} \hat{w}_j^{ls}$$

In the PCA axes, the ridge coefficients are just scaled LS coefficients!

The coefficients that correspond to smaller input variance directions are scaled down more.



Ridge Regression

In the following simulations, quantities are plotted versus the quantity

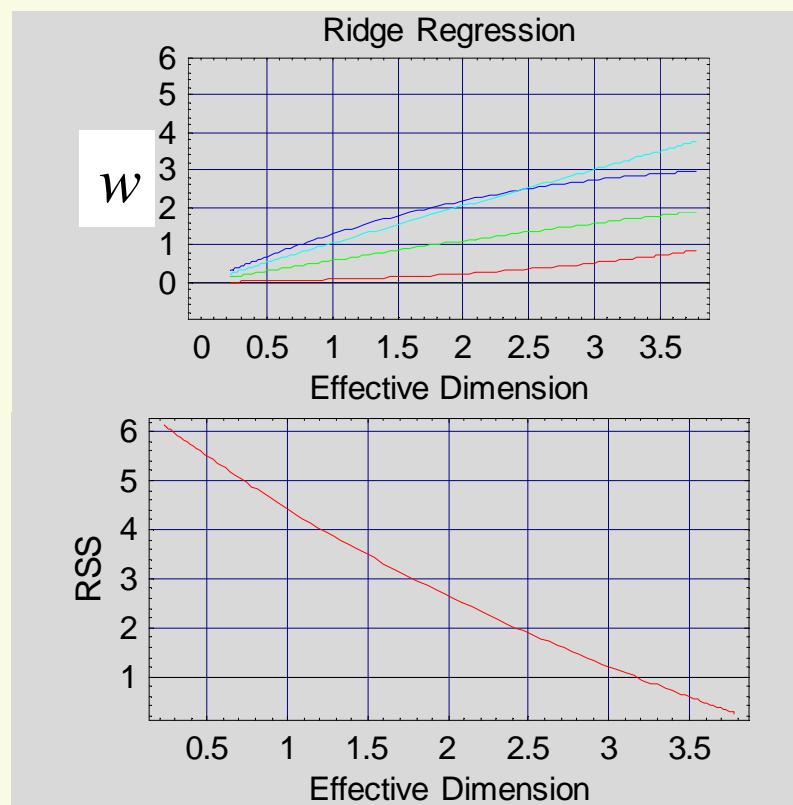
$$df(\lambda) = \sum_{j=1}^M \frac{d_j^2}{d_j^2 + \lambda}$$

This monotonic decreasing function is the *effective degrees of freedom* of the ridge regression fit.

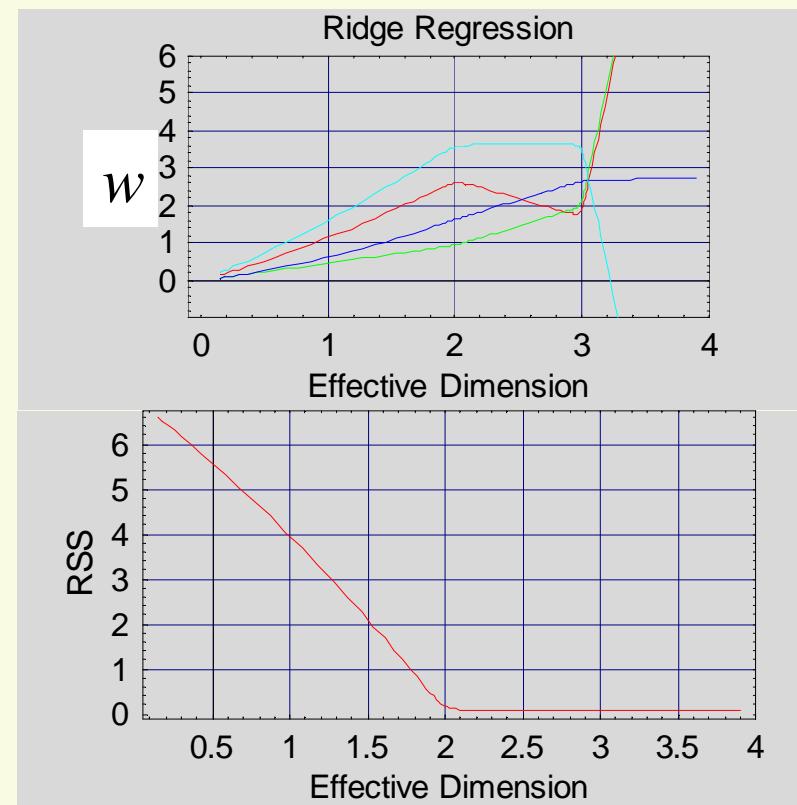
Ridge Regression

Simulation results: 4 dimensions ($w=1,2,3,4$)

No correlation



Strong correlation



Ridge regression is MAP with Gaussian prior

$$\begin{aligned} J(w) &= -\log P(D \mid w)P(w) \\ &= -\log \left[\prod_{i=1}^n N(y_i \mid w^t x_i, \sigma^2) N(w \mid 0, \tau^2) \right] \\ &= \frac{1}{2\sigma^2} (y - Xw)^t (y - Xw) + \frac{1}{2\tau^2} w^t w + const \end{aligned}$$

This is the same objective function that ridge solves, using $\lambda = \sigma^2 / \tau^2$

Ridge: $J(w) = (y - Xw)^t (y - Xw) + \lambda w^t w$

Lasso

Lasso is a shrinkage method like ridge, with a subtle but important difference. It is defined by

$$\hat{w}^{lasso} = \arg \min_{\beta} \left\{ \sum_{i=1}^N \left(y_i - \sum_{j=1}^M x_{ij} w_j \right)^2 \right\}$$

$$\text{subject to } \sum_{j=1}^M |w_j| \leq t$$

There is a similarity to the ridge regression problem: the L_2 ridge regression penalty is replaced by the L_1 lasso penalty.

The L_1 penalty makes the solution non-linear in y and requires a quadratic programming algorithm to compute it.

Lasso

If t is chosen to be larger than t_0 :

$$t \geq t_0 \equiv \sum_{j=1}^M |\hat{w}^{ls}|$$

then the lasso estimation is identical to the least squares. On the other hand, for say $t=t_0/2$, the least squares coefficients are shrunk by about 50% on average.

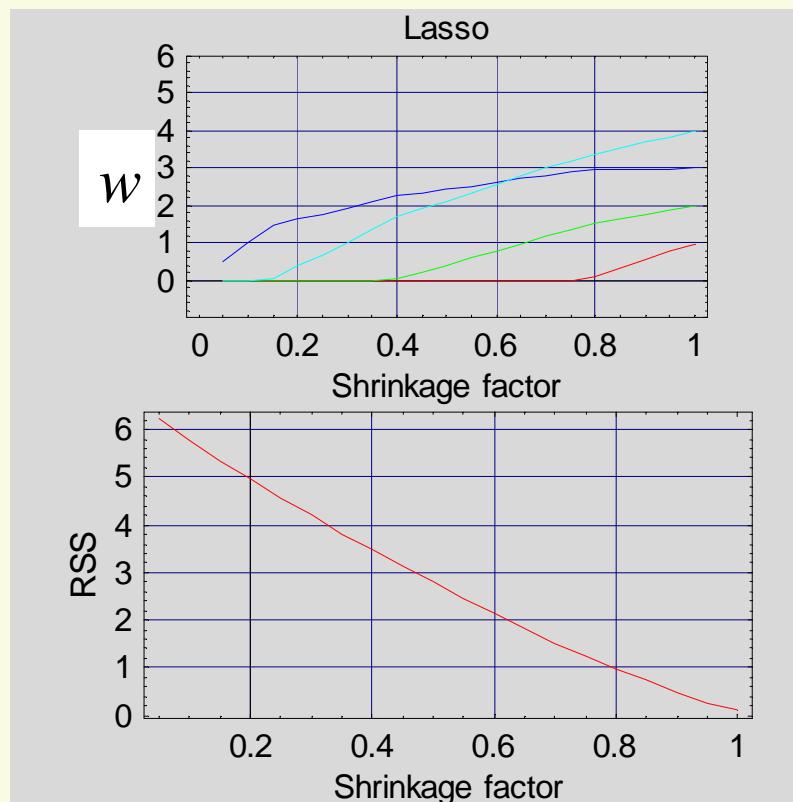
For convenience we will plot the simulation results versus shrinkage factor s :

$$s \equiv \frac{t}{t_0} = \frac{t}{\sum_{j=1}^M |\hat{w}^{ls}|}$$

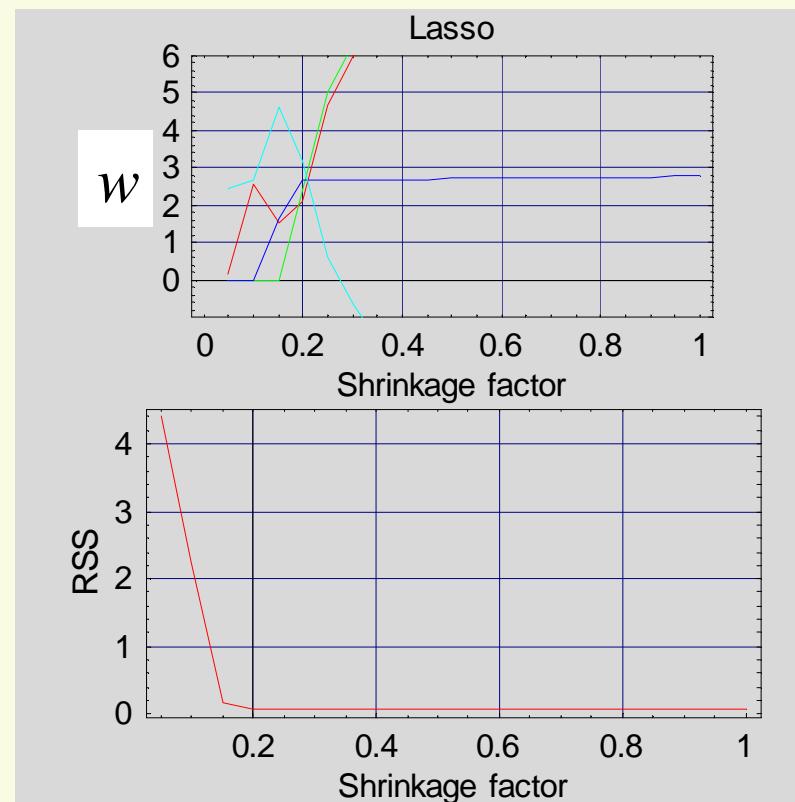
Lasso

Simulation results:

No correlation

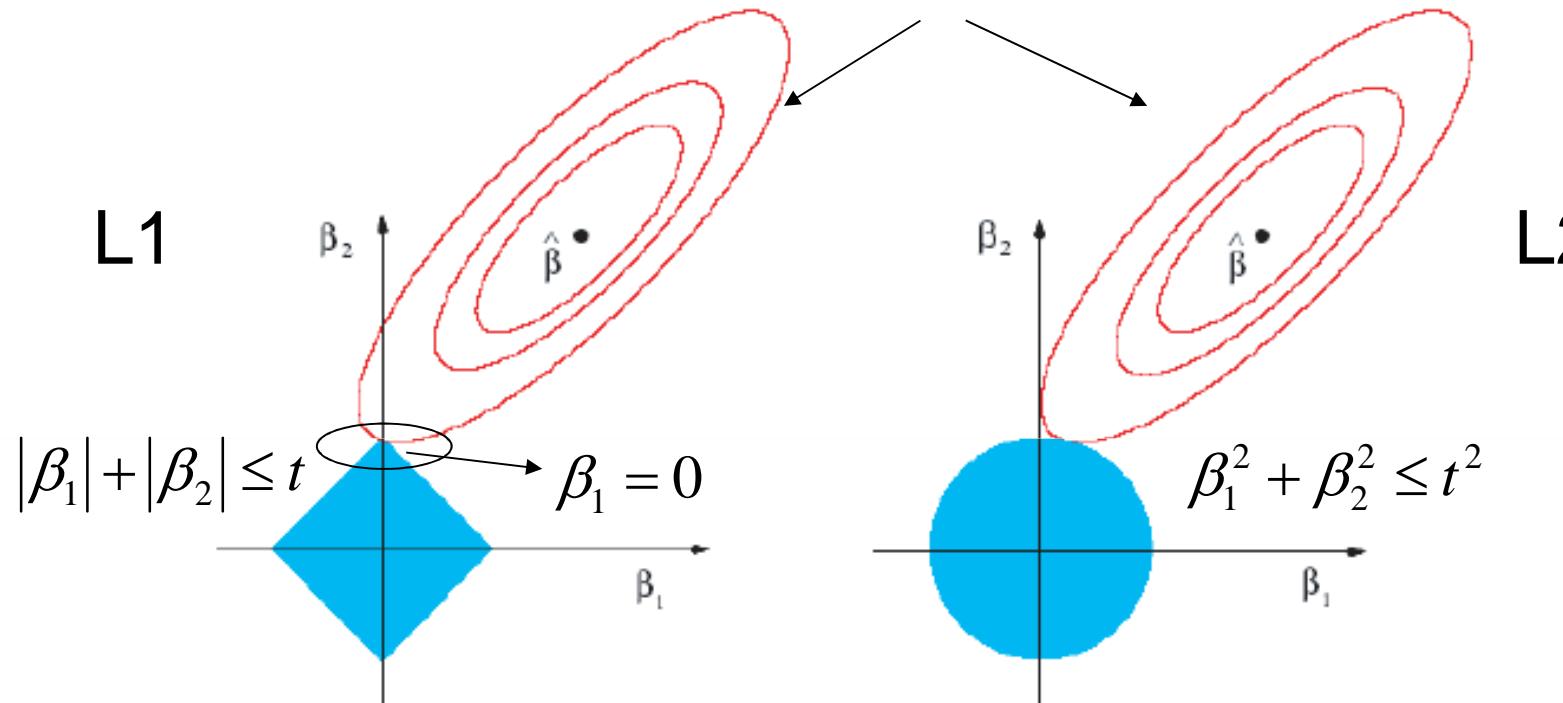


Strong correlation



L2 vs L1 penalties

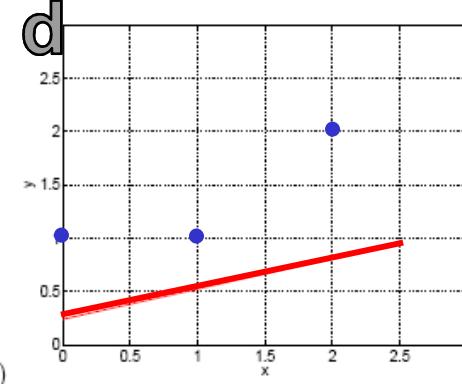
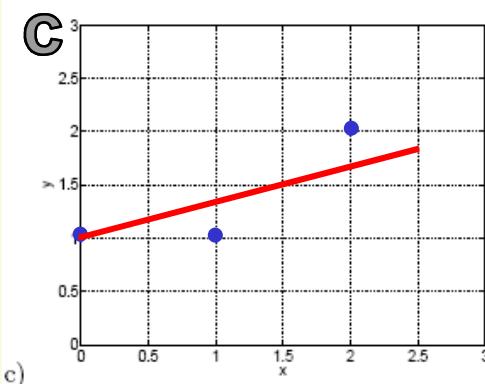
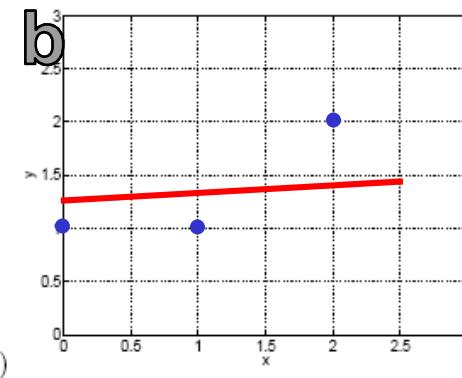
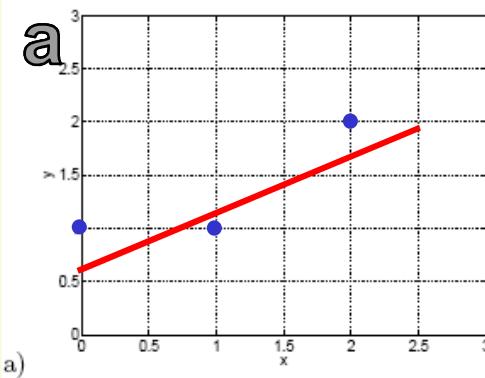
Contours of the LS error function



In Lasso the constraint region has corners;
when the solution hits a corner the
corresponding coefficients becomes 0 (when
 $M > 2$ more than one).

Problem:

Figure 1 plots linear regression results on the basis of only three data points. We used various types of regularization to obtain the plots (see below) but got confused about which plot corresponds to which regularization method. Please assign each plot to one (and only one) of the following regularization method.



C $\sum_{t=1}^3 (y_t - \theta x_t - \theta_0)^2 + \lambda \theta^2$ where $\lambda = 1$

b $\sum_{t=1}^3 (y_t - \theta x_t - \theta_0)^2 + \lambda \theta^2$ where $\lambda = 10$

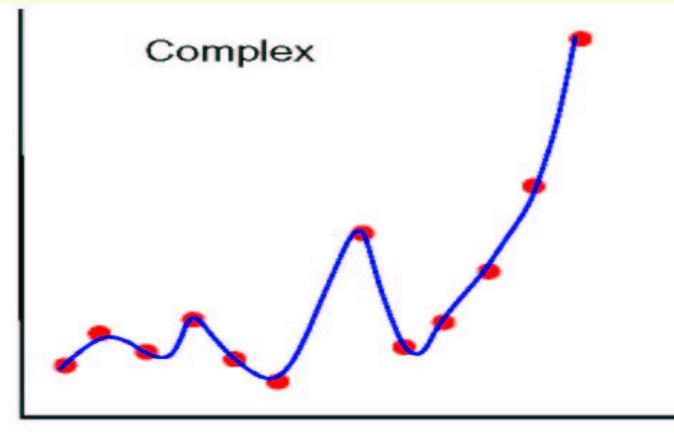
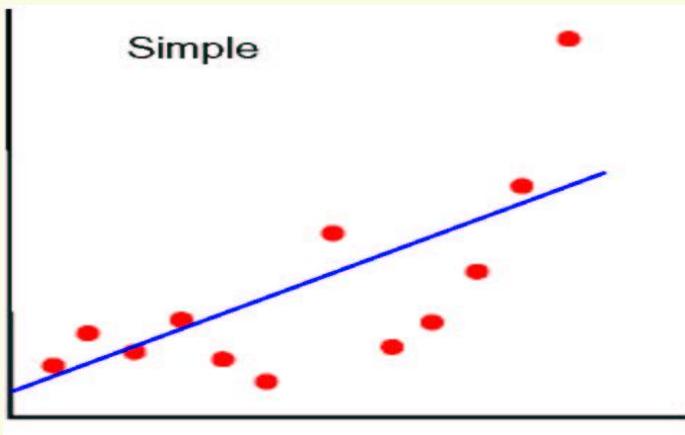
a $\sum_{t=1}^3 (y_t - \theta x_t - \theta_0)^2 + \lambda(\theta^2 + \theta_0^2)$ where $\lambda = 1$

d $\sum_{t=1}^3 (y_t - \theta x_t - \theta_0)^2 + \lambda(\theta^2 + \theta_0^2)$ where $\lambda = 10$

Bias-Variance Decomposition
Error Estimators
Cross-Validation

Bias-Variance tradeoff – Intuition

- Model too “simple” → does not fit the data well – a biased solution.



- Model too complex → small changes to the data, solution changes a lot – high-variance solution.

Expected Loss

- Let $h(x) = E[t|x]$ be the optimal predictor and $y(x)$ our actual predictor, which will incur the following expected loss

$$\begin{aligned} E(y(x)-t)^2 &= \iint (y(x)-h(x)+h(x)-t)^2 p(x,t) dx dt \\ &= E\left((y(x)-h(x))^2 + 2(y(x)-h(x))(h(x)-t) + (h(x)-t)^2\right) \\ &= \int (y(x)-h(x))^2 p(x) dx + \iint (h(x)-t)^2 p(x) dx dt \end{aligned}$$

since $\int (E[t|x]-t)p(t|x)dt = 0$

$$E(y(x)-t)^2 = \int (y(x)-h(x))^2 p(x) dx + \iint (h(x)-t)^2 p(x) dt$$

Noise term

Source of error 1

Sources of error 1 –noise

- What if we have perfect learner, infinite data?
 - Our learning solution $y(x)$ satisfies $y(x)=h(x)$
 - Still have remaining, *unavoidable error* of σ^2 due to noise ε

$$\int \int \underbrace{(y(x) - t)^2}_{\begin{array}{c} \uparrow \\ h(x) \\ \hline h(x) + \varepsilon \end{array}} p(f(x) = t \mid x) p(x) dt dx$$
$$\varepsilon^2, \quad \varepsilon \sim N(0, \sigma^2)$$
$$= E[\varepsilon^2] = \sigma^2$$

Bias-variance decomposition

- Focus on $\int (y(x) - h(x))^2 p(x) dx$
- Let us first examine expected loss averaging over data sets.
- For one data set D and one test point x:

$$\begin{aligned}(y(x, D) - h(x))^2 &= (y(x, D) - E_D[y(x, D)] + E_D[y(x, D)] - h(x))^2 \\&= (y(x, D) - E_D[y(x, D)])^2 + (E_D[y(x, D)] - h(x))^2 \\&\quad + \boxed{2(y(x, D) - E_D[y(x, D)])(E_D[y(x, D)] - h(x))} \rightarrow E_D \text{ of this} \\&\quad \text{cancels to zero}\end{aligned}$$

- Hence

$$\begin{aligned}E_D(y(x, D) - h(x))^2 &\quad \downarrow \quad \downarrow \\&= (E_D[y(x, D)] - h(x))^2 + E_D(y(x, D) - E_D[y(x, D)])^2 \\&= \text{bias}^2 + \text{variance}\end{aligned}$$

Bias

- Suppose you are given a dataset D with m samples from some distribution.
- You learn function $y(x)$ from data D
- If you sample a different datasets, you will learn different $y(x)$
- **Expected hypothesis:** $E_D[y(x,D)]$
- **Bias:** difference between what you expect to learn and the truth.
 - Measures how well you expect to represent true solution
 - Decreases with more complex model

$$\text{bias}^2 = \int_x (E_D[y(x,D)] - h(x))^2 p(x) dx$$

x Expected
 to learn True model

Variance

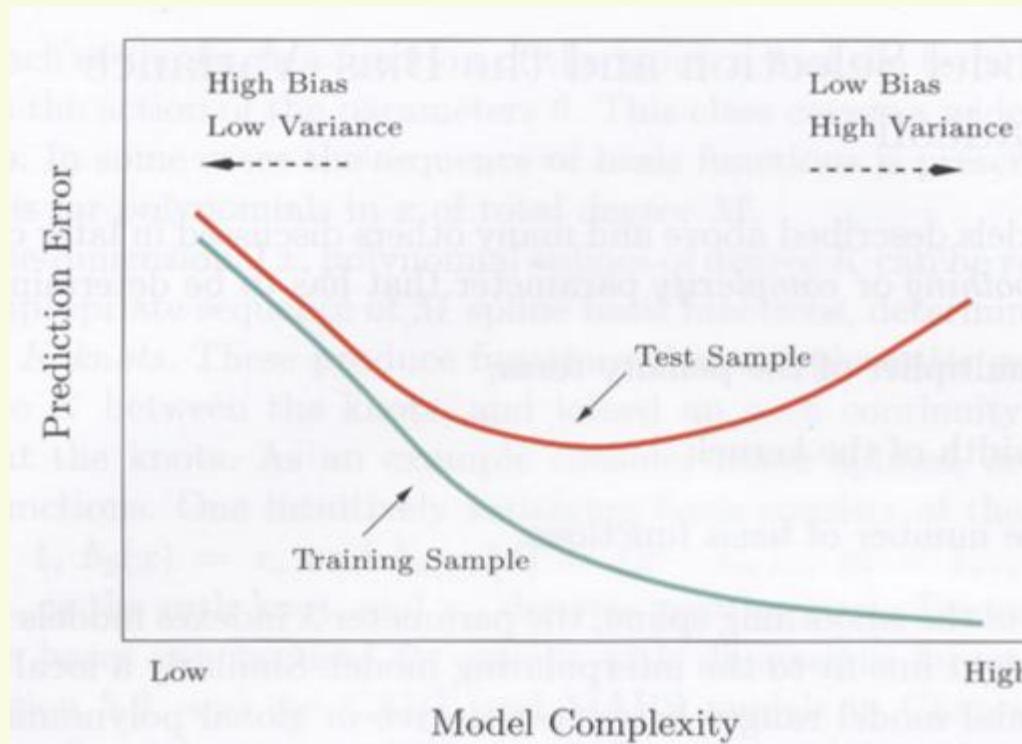
- **Variance:** difference between what you expect to learn and what you learn from a particular dataset
 - Measures how sensitive learner is to a specific dataset
 - Decreases with simpler model

$$\text{variance} = \int_x E_D \left(\left(\underset{\uparrow}{y(x, D)} - \underset{\nwarrow}{E_D[y(x, D)]} \right)^2 \right) p(x) dx$$

Learned for D **Expected to learn,
averaged over datasets**

Bias-Variance Tradeoff

- Choice of hypothesis class introduces learning bias
- More complex class → less bias
- More complex class → more variance



The Expected Prediction Error

- The expected prediction squared error over fixed size training sets D drawn from $P(X,T)$ can be expressed as sum of three components:

unavoidable error + bias² + variance

where

$$\text{unavoidable error} = \sigma^2$$

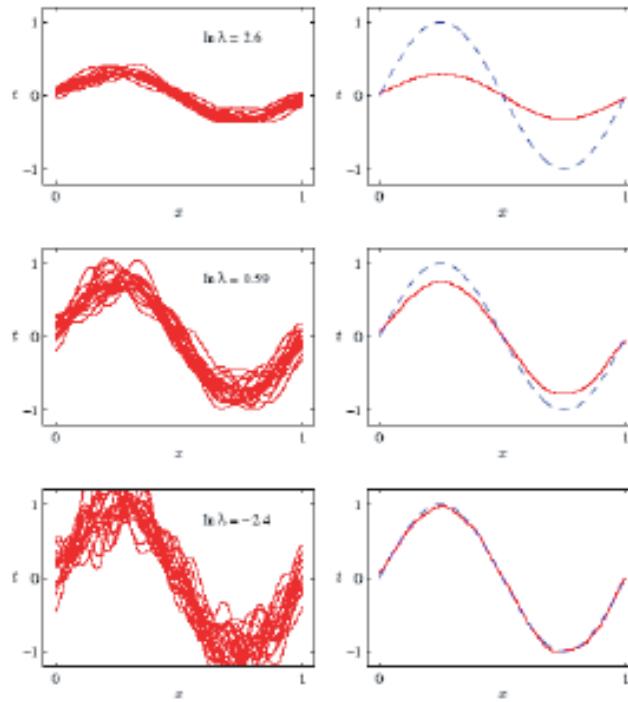
$$\text{bias}^2 = \int_x (E_D[y(x)] - h(x))^2 p(x) dx$$

$$\text{variance} = \int E_D [(y(x) - E_D[y(x)])^2] p(x) dx$$

AVERAGE OVER TRAINING SETS

Higher variance

Lower bias



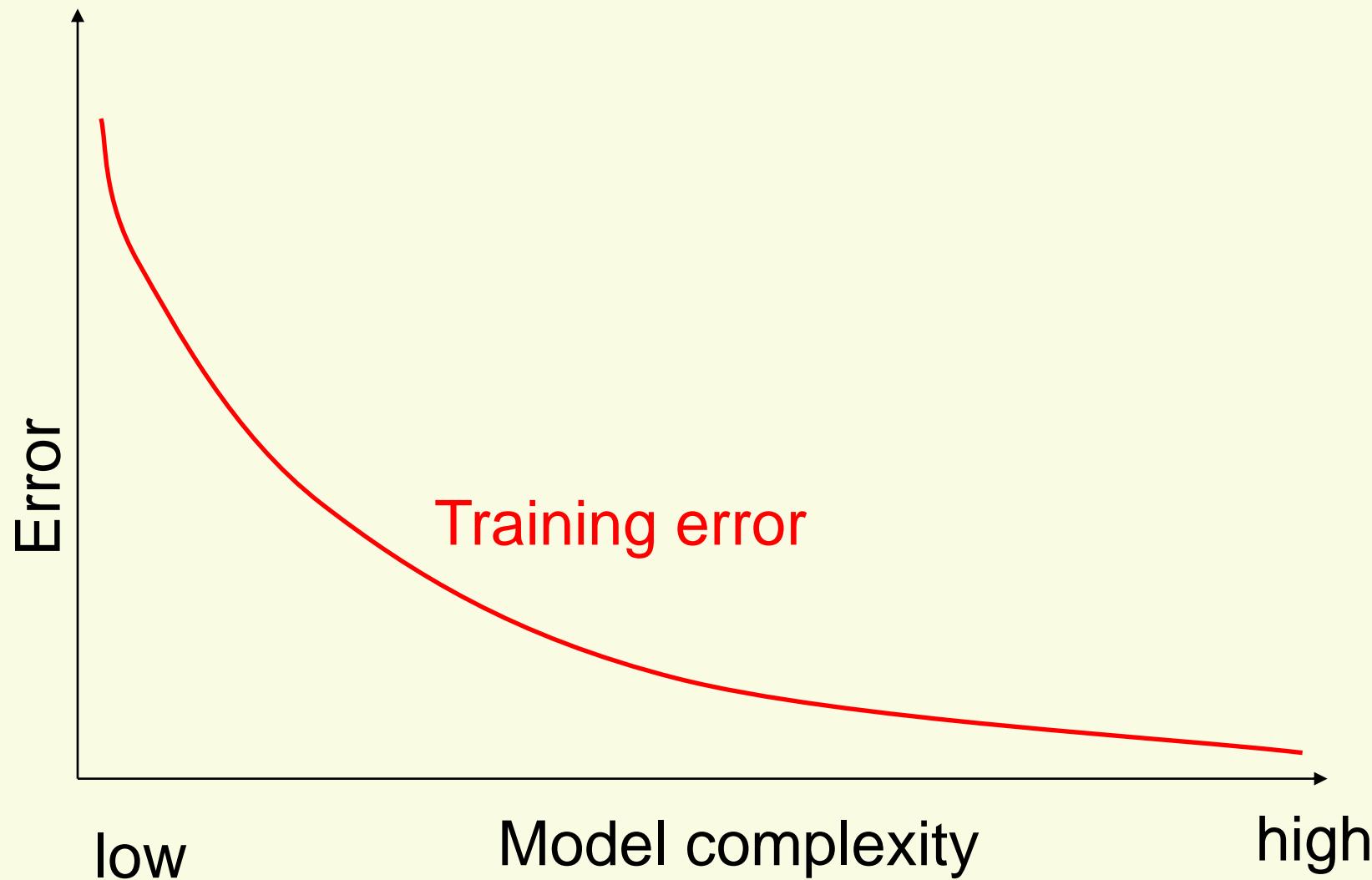
Highly regularized have low variance but high bias.

Training Error

- Given a training data, choose a loss function. (e.g., squared error (L2) for regression)
- **Training set error:** For a particular set of parameters, loss function on training data:

$$Err_{train}(w) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \left(t(x_i) - \sum_{j=1}^M w_j y(x_j) \right)^2$$

Training Error vs. Model Complexity



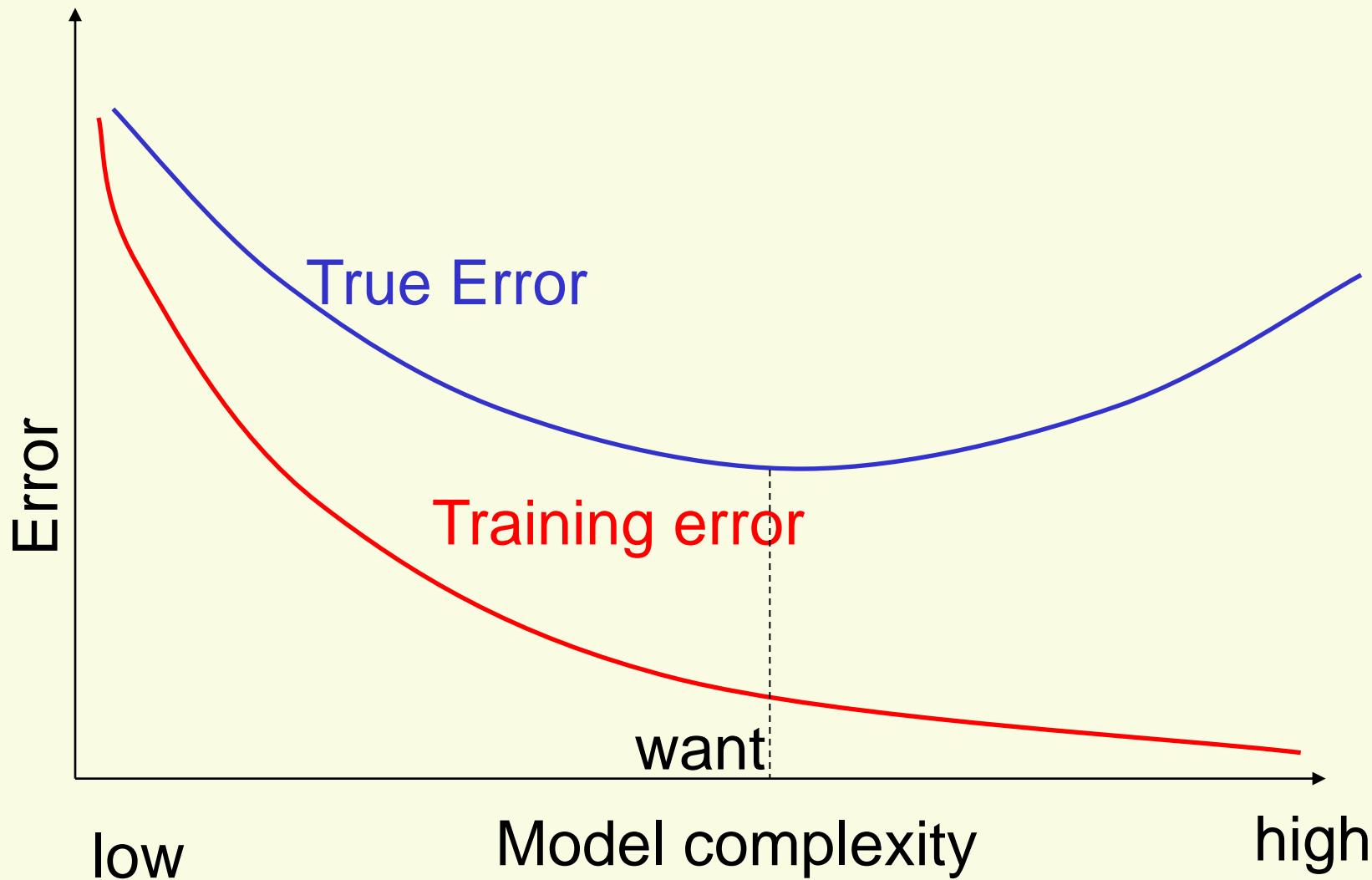
Prediction (Generalization) Error

- Training set error can be poor measure of "quality" of solution
- **Prediction error:** We really care about error over all possible input points, not just training data:

$$Err_{true}(w) = E_X \left[\left(t(x) - \sum_{j=1}^M w_i y(x_j) \right)^2 \right]$$

$$= \int_X \left(t - \sum_{j=1}^M w_i y(x_j) \right)^2 dx$$

Prediction Error vs. Model Complexity



Why training set error doesn't approximate prediction error?

- Sampling approximation of prediction error:

$$Err_{true}(w) \approx \frac{1}{p} \sum_{i=1}^p \left(t - \sum_{j=1}^M w_i y(x_j) \right)^2$$

- Training error

$$Err_{train}(w) = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} \left(t(x_i) - \sum_{j=1}^M w_i y(x_j) \right)^2$$

- Very similar equations!!!

Why is training set a bad measure of prediction error???

Why is training set a bad measure of prediction error???

- Training error is a good estimate for a single \mathbf{w} ,
- But we optimized \mathbf{w} with respect to the training error, and found \mathbf{w} that is good for this set of samples.
- **Training error is a (optimistically) biased estimate of prediction error**

Test Error

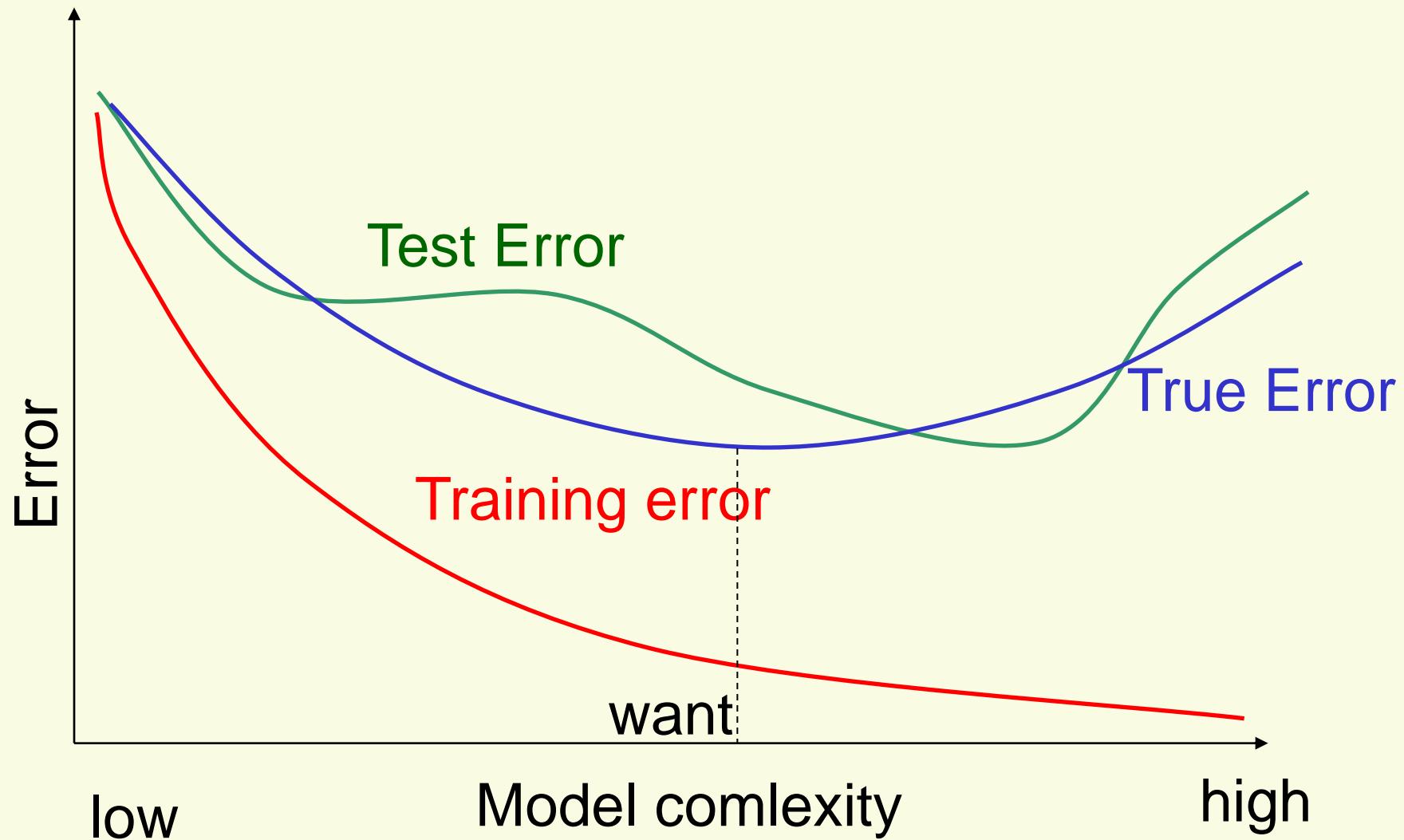
- Given a dataset, **randomly** split it into two parts:
 - Training data – $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_{\text{train}}}\}$
 - Test data – $\{\mathbf{x}_1, \dots, \mathbf{x}_{N_{\text{test}}}\}$
- Use training data to optimize parameters \mathbf{w}
- **Test set error:** For the ***final solution*** \mathbf{w}^* , evaluate the error using:

$$Err_{\text{test}}(\mathbf{w}^*) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left(t(x_i) - \sum_{j=1}^M w_j^* y(x_j) \right)^2$$

Unbiased but has variance

Test set only unbiased if you never do any learning on the test data
For example, if you use the test set to select the degree of the polynomial...no longer unbiased!!!

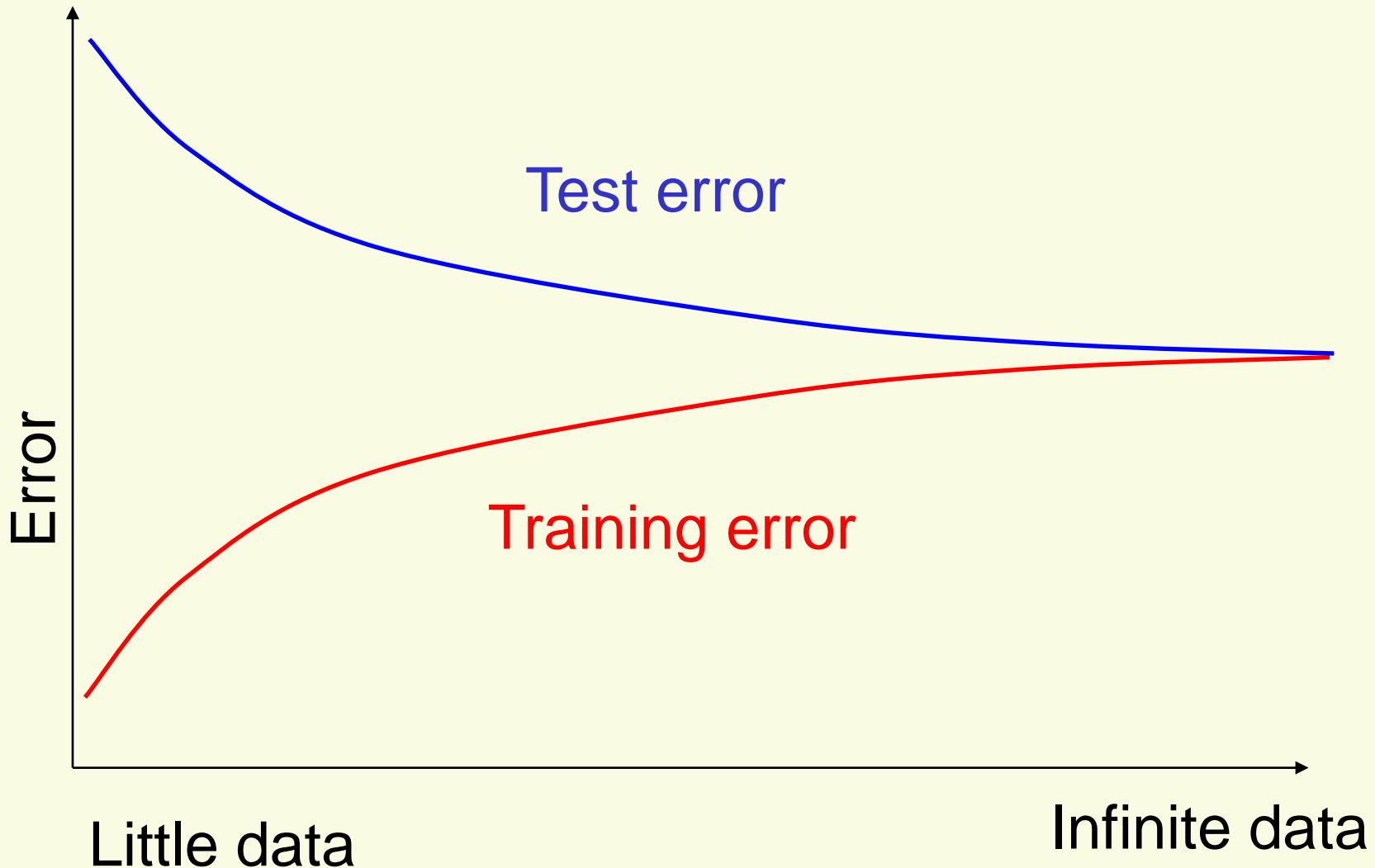
Test Set Error vs. Model Complexity



How many points to use for training/testing?

- Very hard question to answer!
 - Too few training points, learned w is bad
 - Too few test points, you never know if you reached a good solution
- Theory proposes error bounds (advanced course)
- Typically:
 - if you have a reasonable amount of data, pick test set “large enough” for a “reasonable” estimate of error, and use the rest for learning
 - if you have little data, then you need to use some special techniques e.g., bootstrapping

Error as a function of number of training examples for a fixed model complexity



Overfitting

- With too few training examples our model may achieve zero training error but nevertheless has a large generalization error
- When the training error no longer bears any relation to the generalization error the model overfits the data

Cross-validation for detecting and preventing overfitting

Note to other teachers and users of these slides. Andrew would be delighted if you found this source material useful in giving your own lectures. Feel free to use these slides verbatim, or to modify them to fit your own needs. PowerPoint originals are available. If you make use of a significant portion of these slides in your own lecture, please include this message, or the following link to the source repository of Andrew's tutorials:
<http://www.cs.cmu.edu/~awm/tutorials>.
Comments and corrections gratefully received.

Andrew W. Moore

Professor

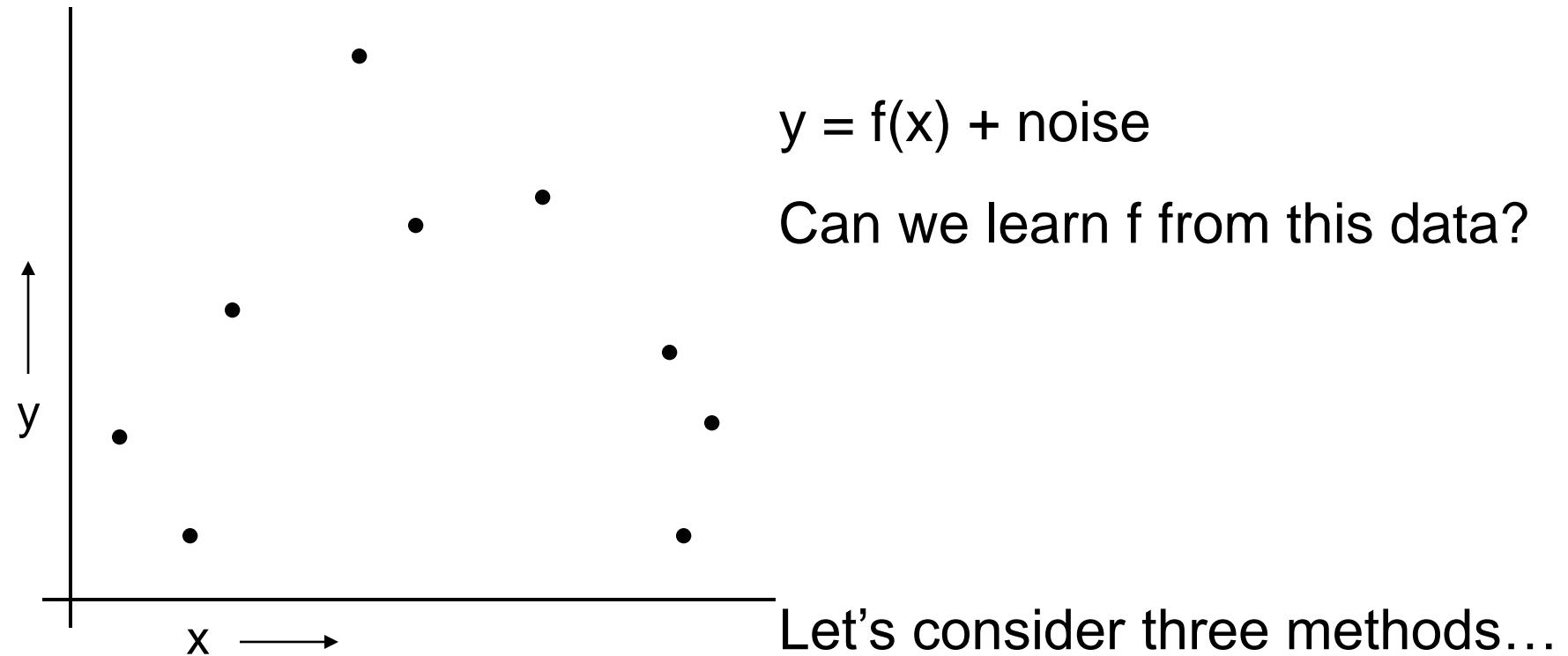
**School of Computer Science
Carnegie Mellon University**

www.cs.cmu.edu/~awm

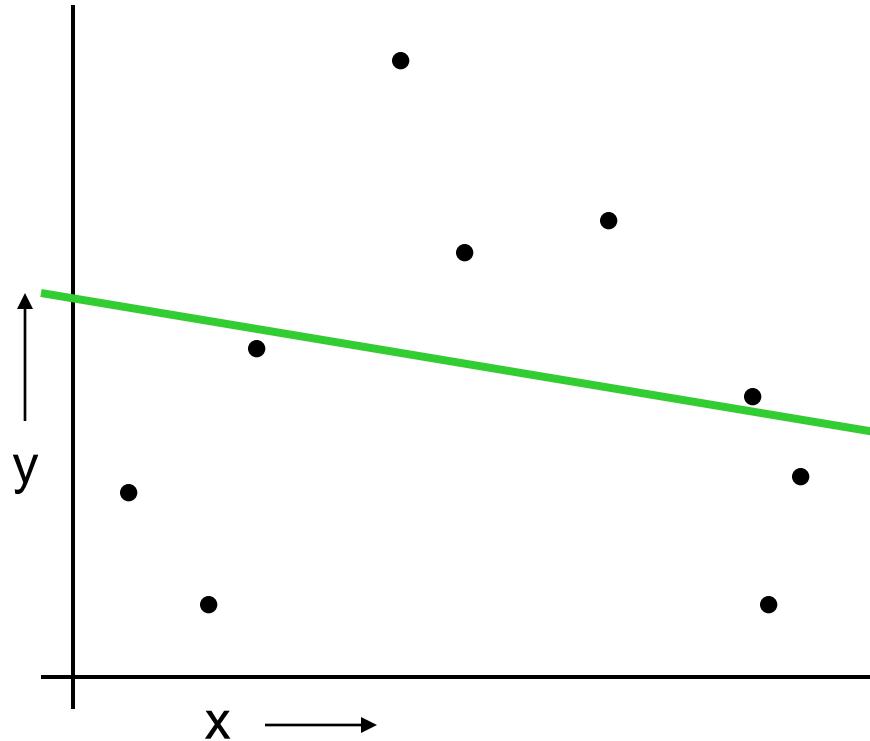
awm@cs.cmu.edu

412-268-7599

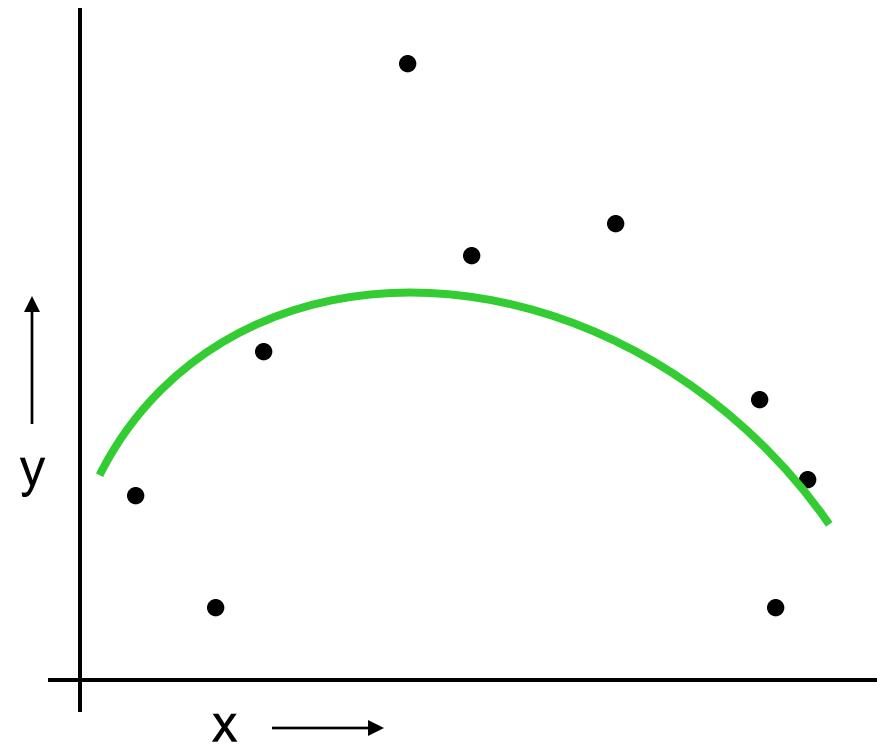
A Regression Problem



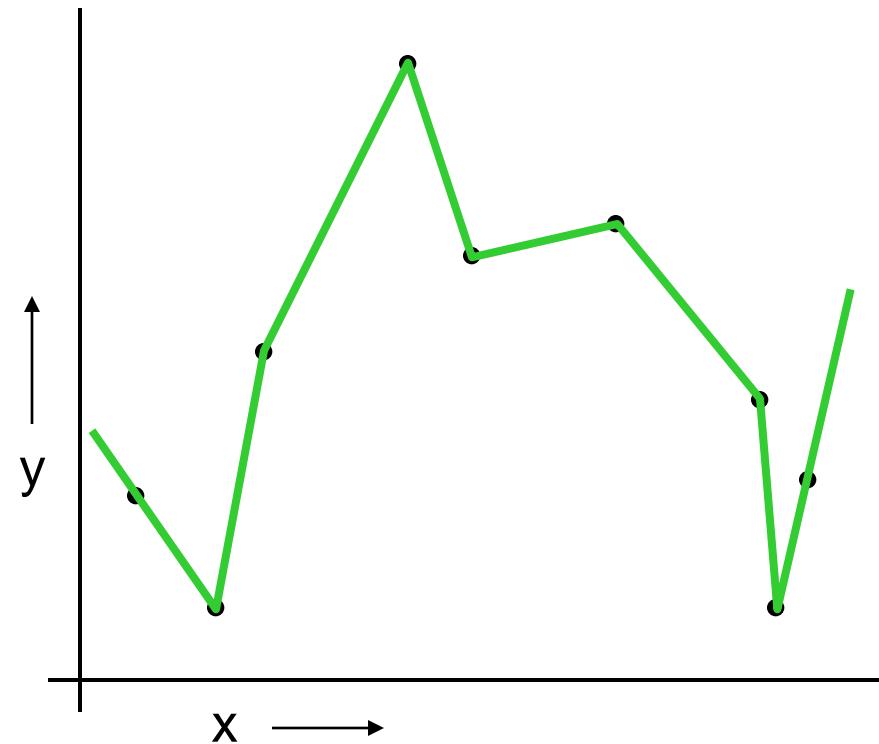
Linear Regression



Quadratic Regression

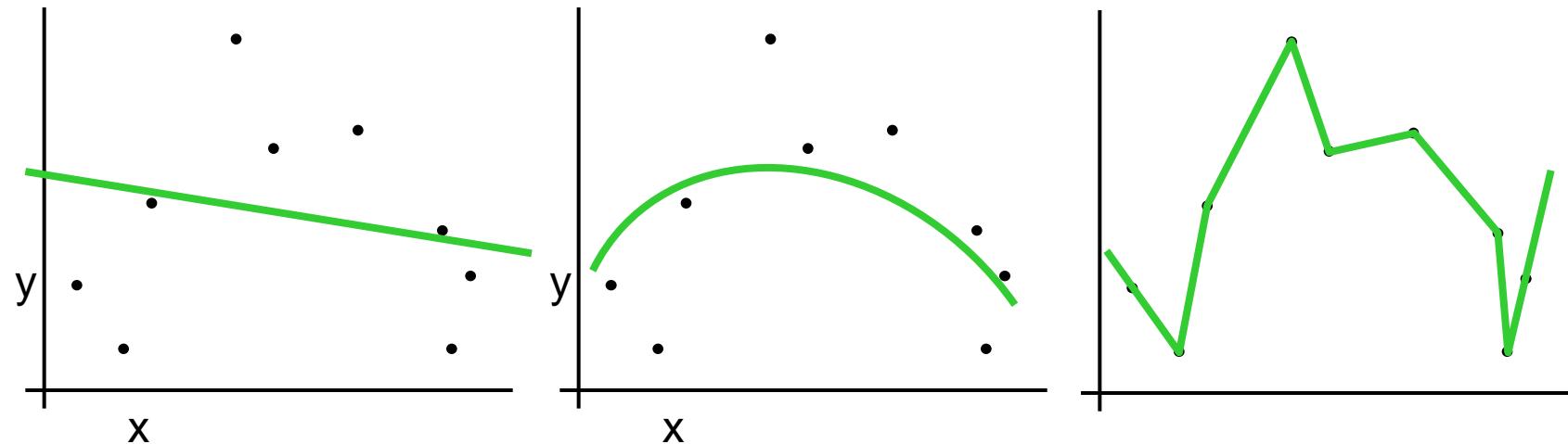


Join-the-dots



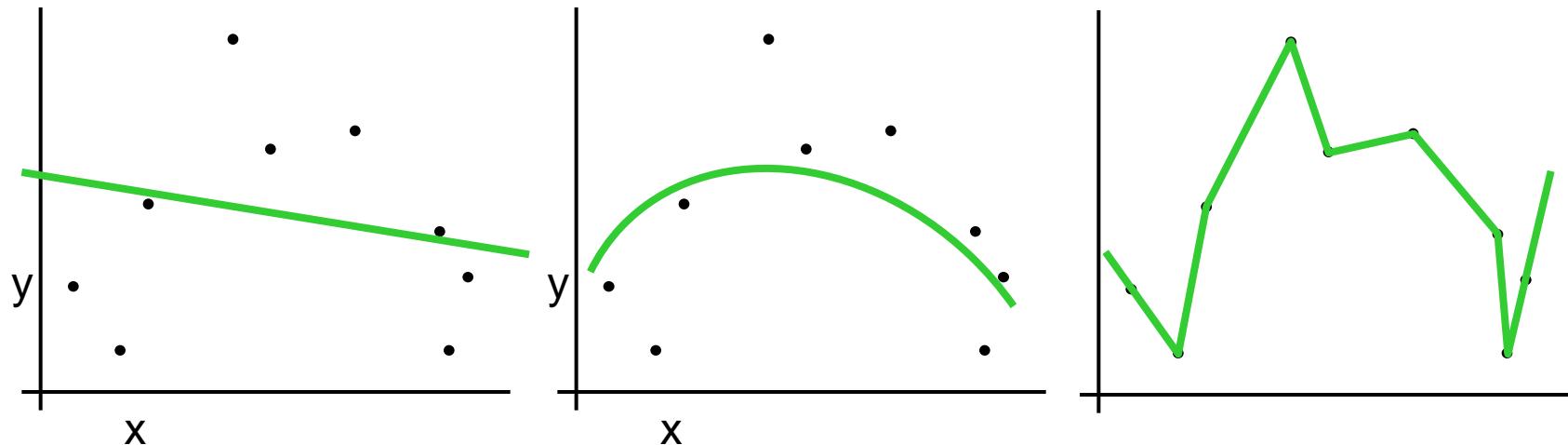
Also known as **piecewise
linear nonparametric
regression** if that makes
you feel better

Which is best?



Why not choose the method with the best fit to the data?

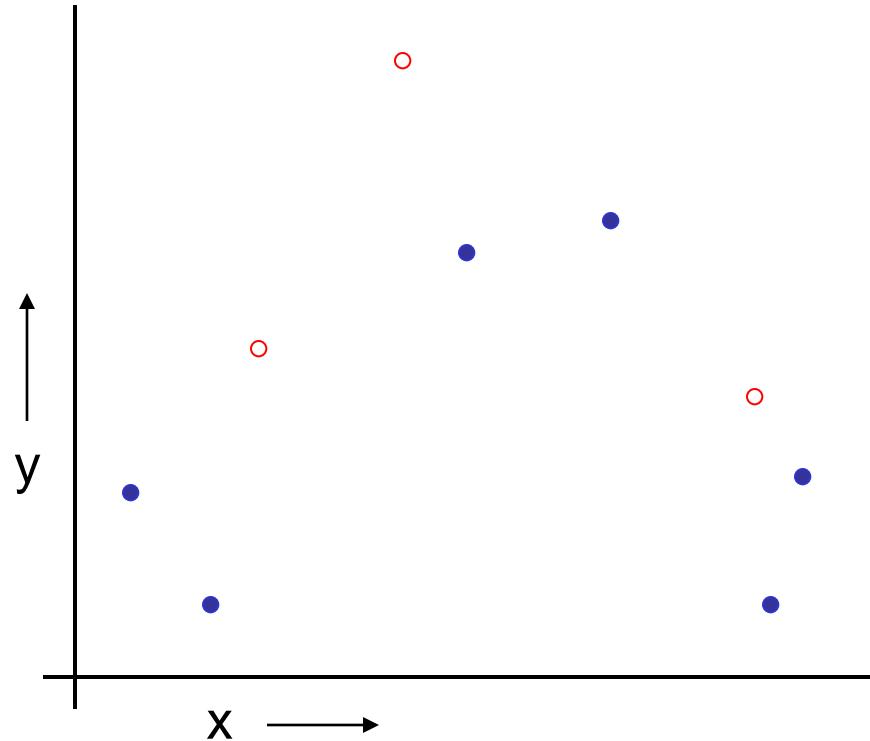
What do we really want?



Why not choose the method with the best fit to the data?

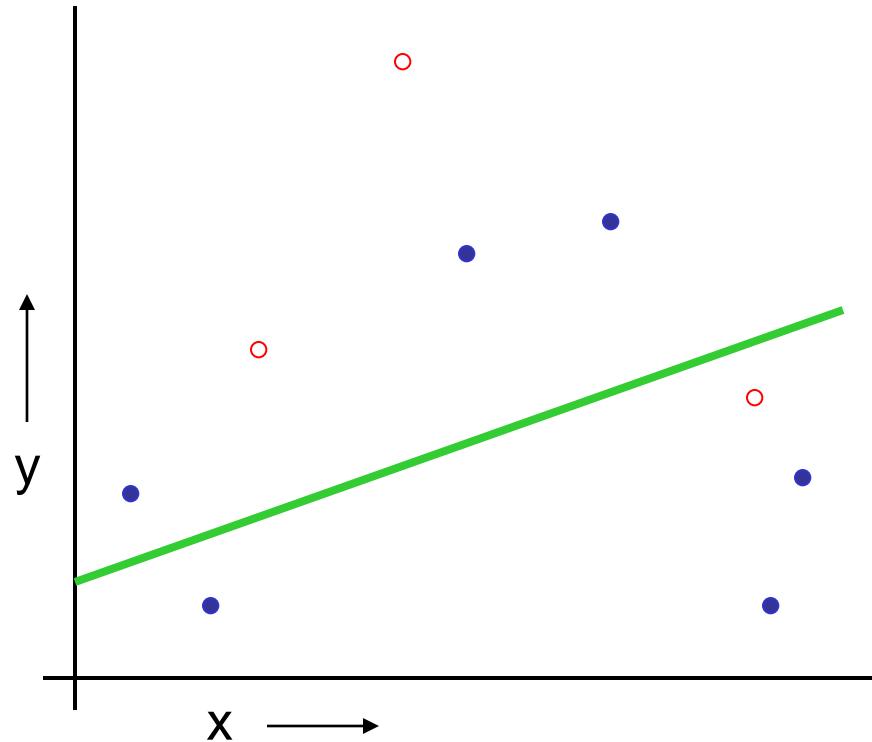
“How well are you going to predict future data drawn from the same distribution?”

The test set method



1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**

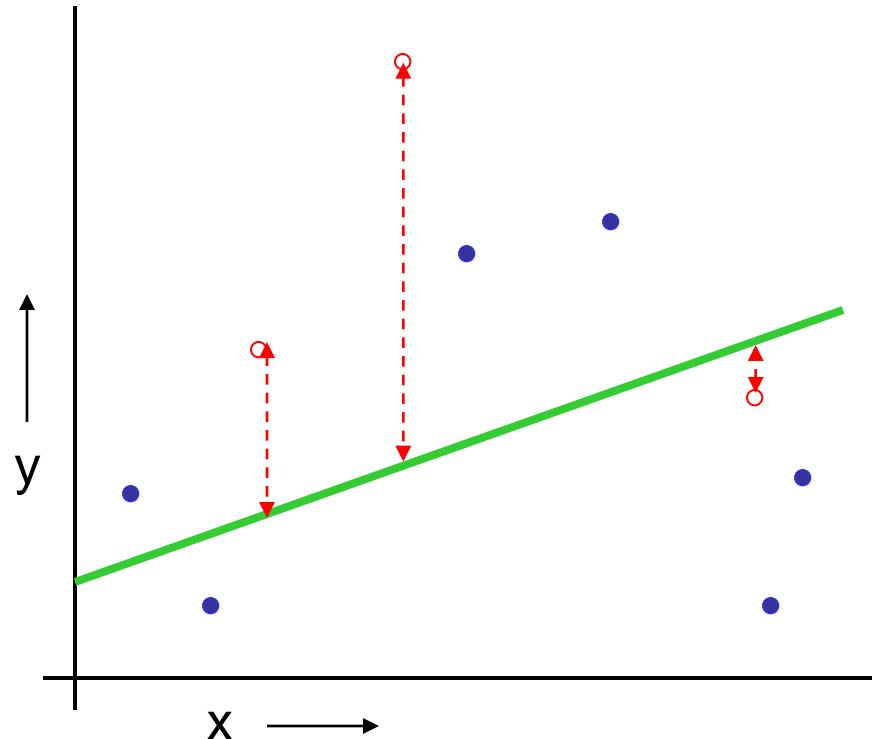
The test set method



1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a training set
3. Perform your regression on the training set

(Linear regression example)

The test set method

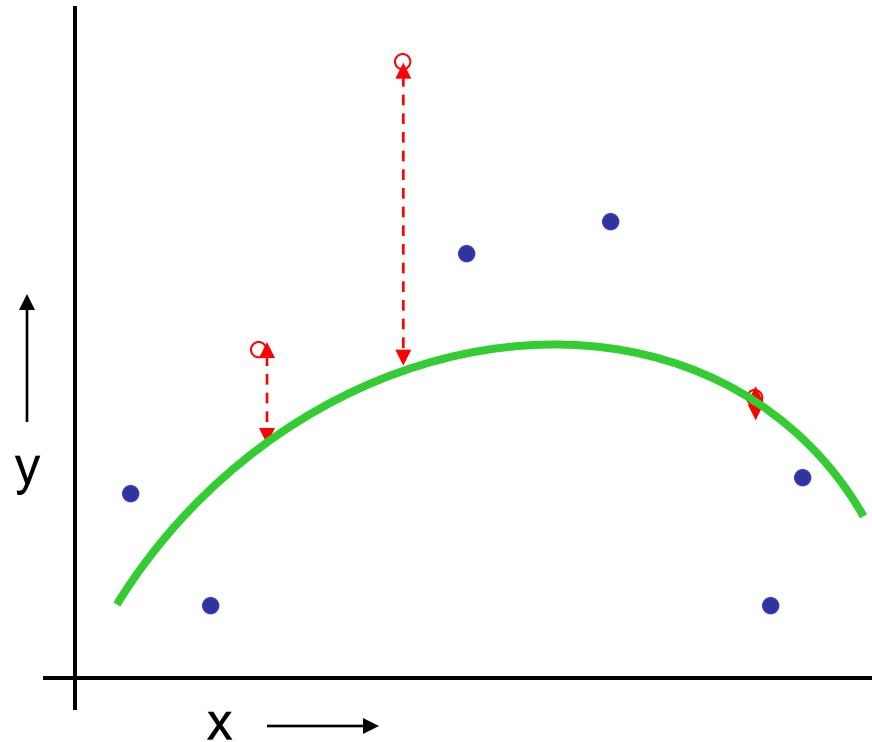


(Linear regression example)

Mean Squared Error = 2.4

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the **training set**
4. Estimate your future performance with the **test set**

The test set method

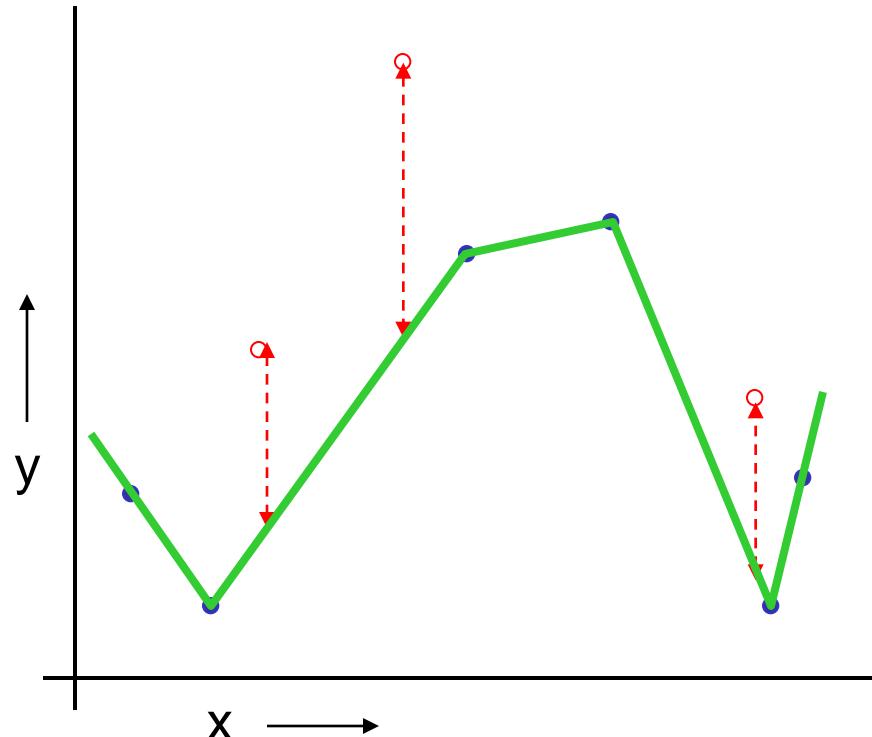


(Quadratic regression example)

Mean Squared Error = 0.9

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set
4. Estimate your future performance with the **test set**

The test set method



(Join the dots example)

Mean Squared Error = 2.2

1. Randomly choose 30% of the data to be in a **test set**
2. The remainder is a **training set**
3. Perform your regression on the training set
4. Estimate your future performance with the test set

The test set method

Good news:

- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:

- What's the downside?

The test set method

Good news:

- Very very simple
- Can then simply choose the method with the best test-set score

Bad news:

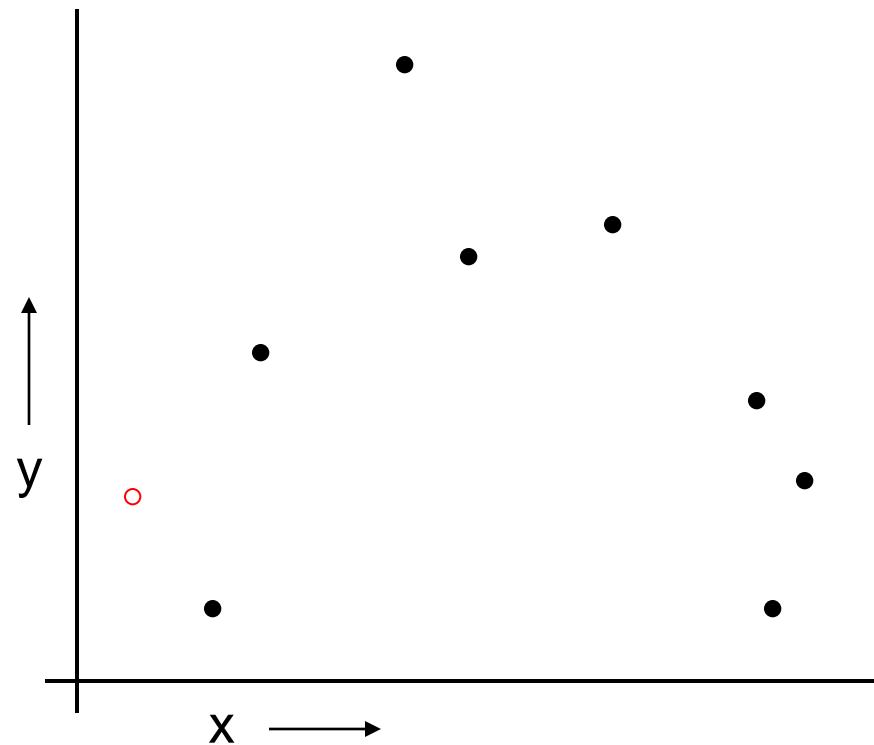
- Wastes data: we get an estimate of the best method to apply to 30% less data
- If we don't have much data, our test-set might just be lucky or unlucky

We say the “test-set estimator of performance has high variance”

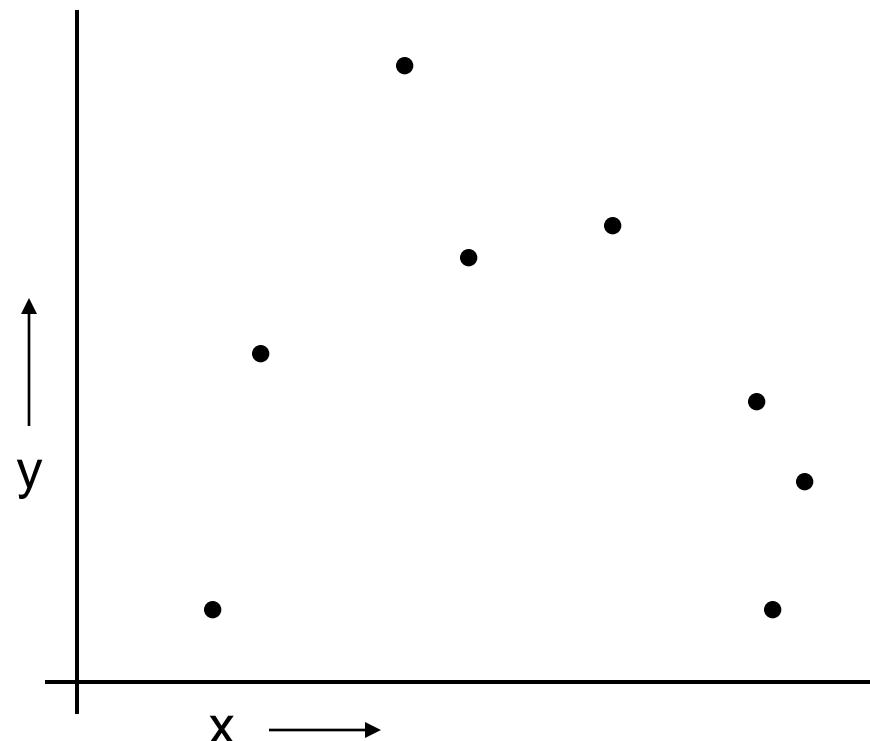
LOOCV (Leave-one-out Cross Validation)

For k=1 to R

1. Let (x_k, y_k) be the k^{th} record



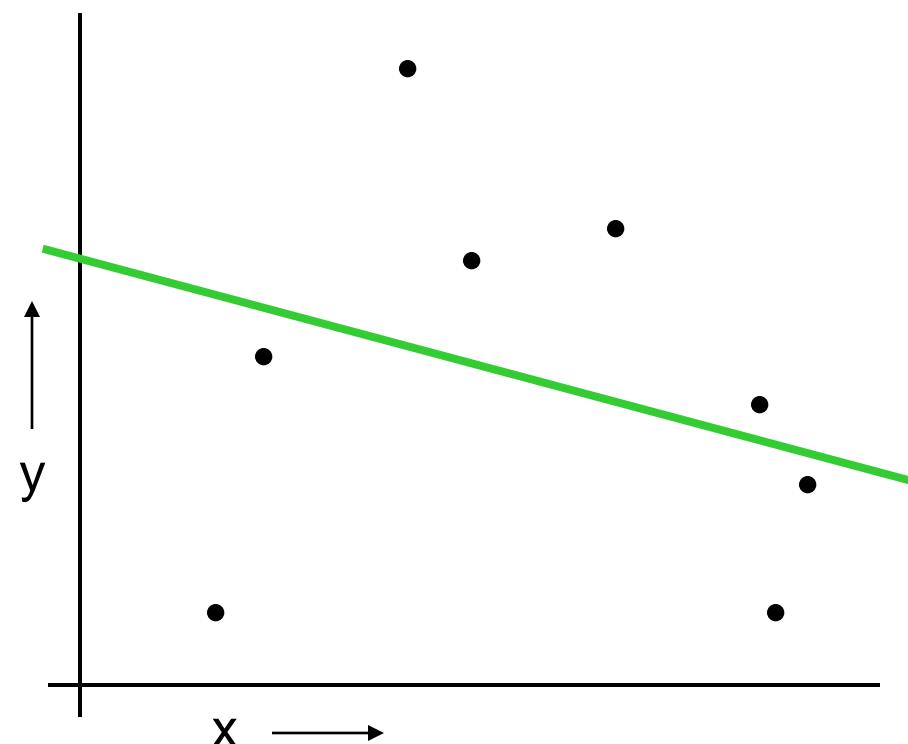
LOOCV (Leave-one-out Cross Validation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset

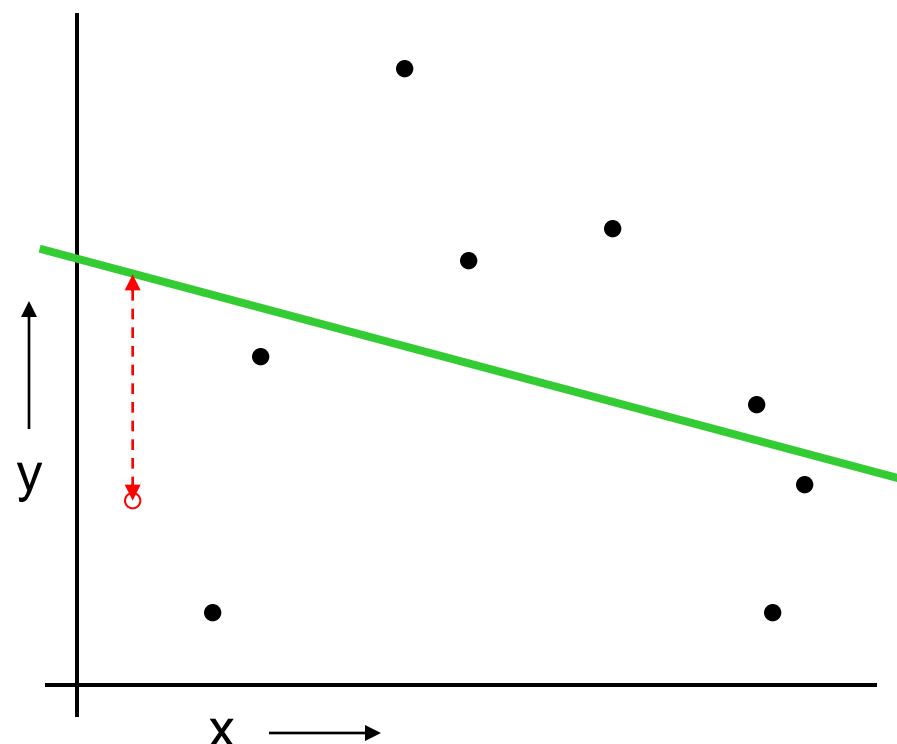
LOOCV (Leave-one-out Cross Validation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints

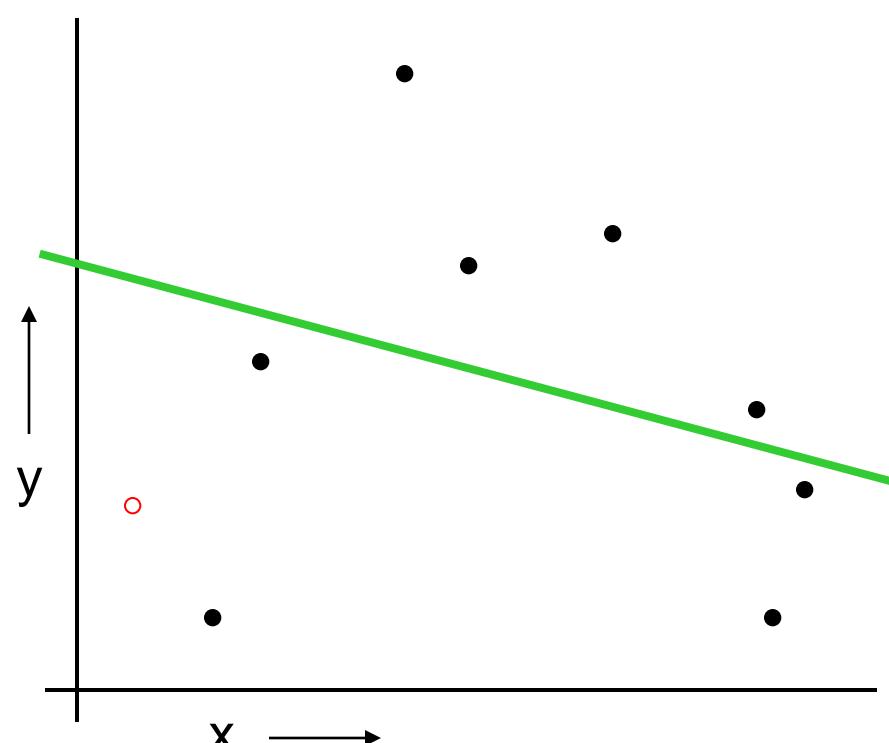
LOOCV (Leave-one-out Cross Validation)



For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining R-1 datapoints
4. Note your error (x_k, y_k)

LOOCV (Leave-one-out Cross Validation)

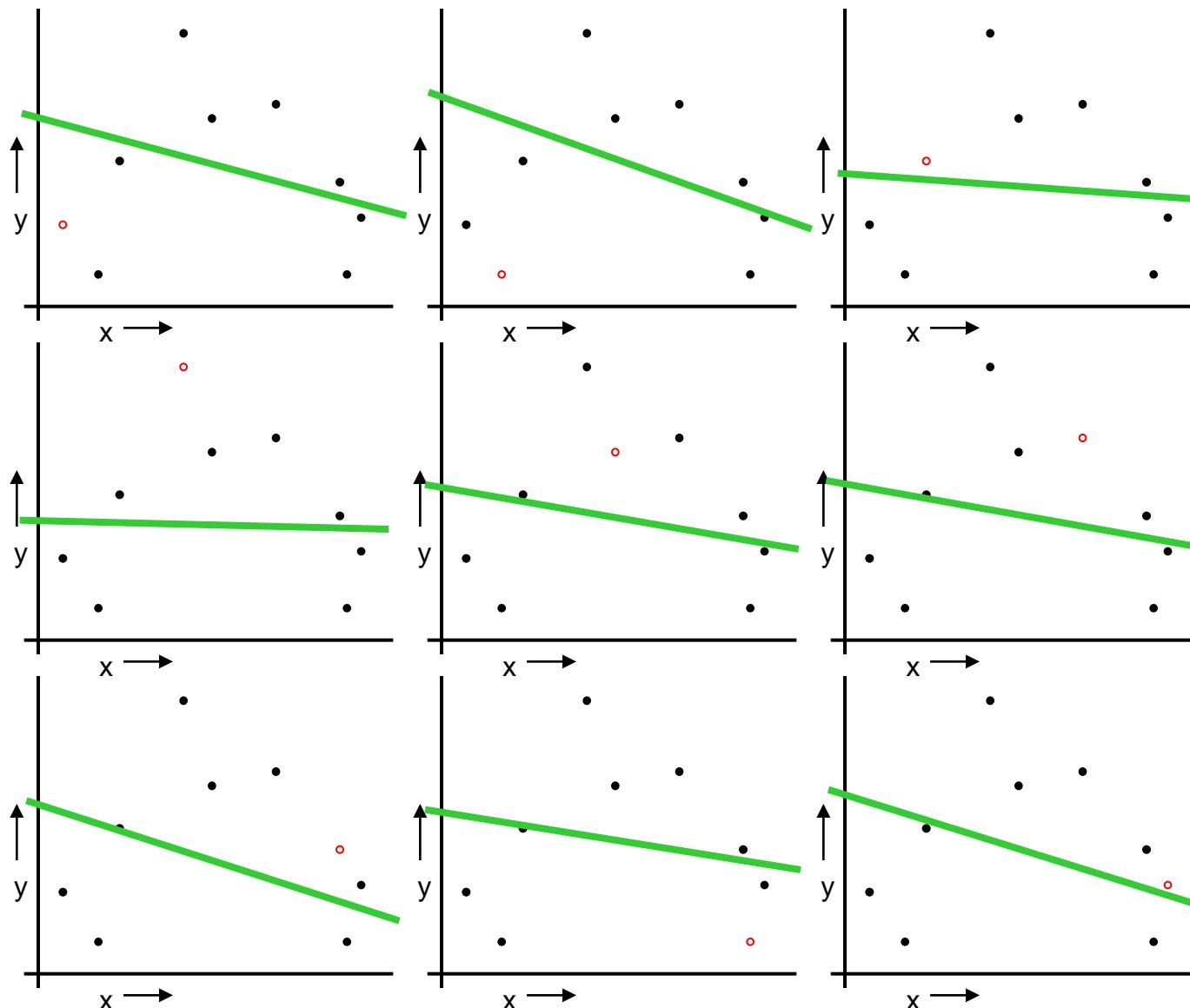


For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points,
report the mean error.

LOOCV (Leave-one-out Cross Validation)



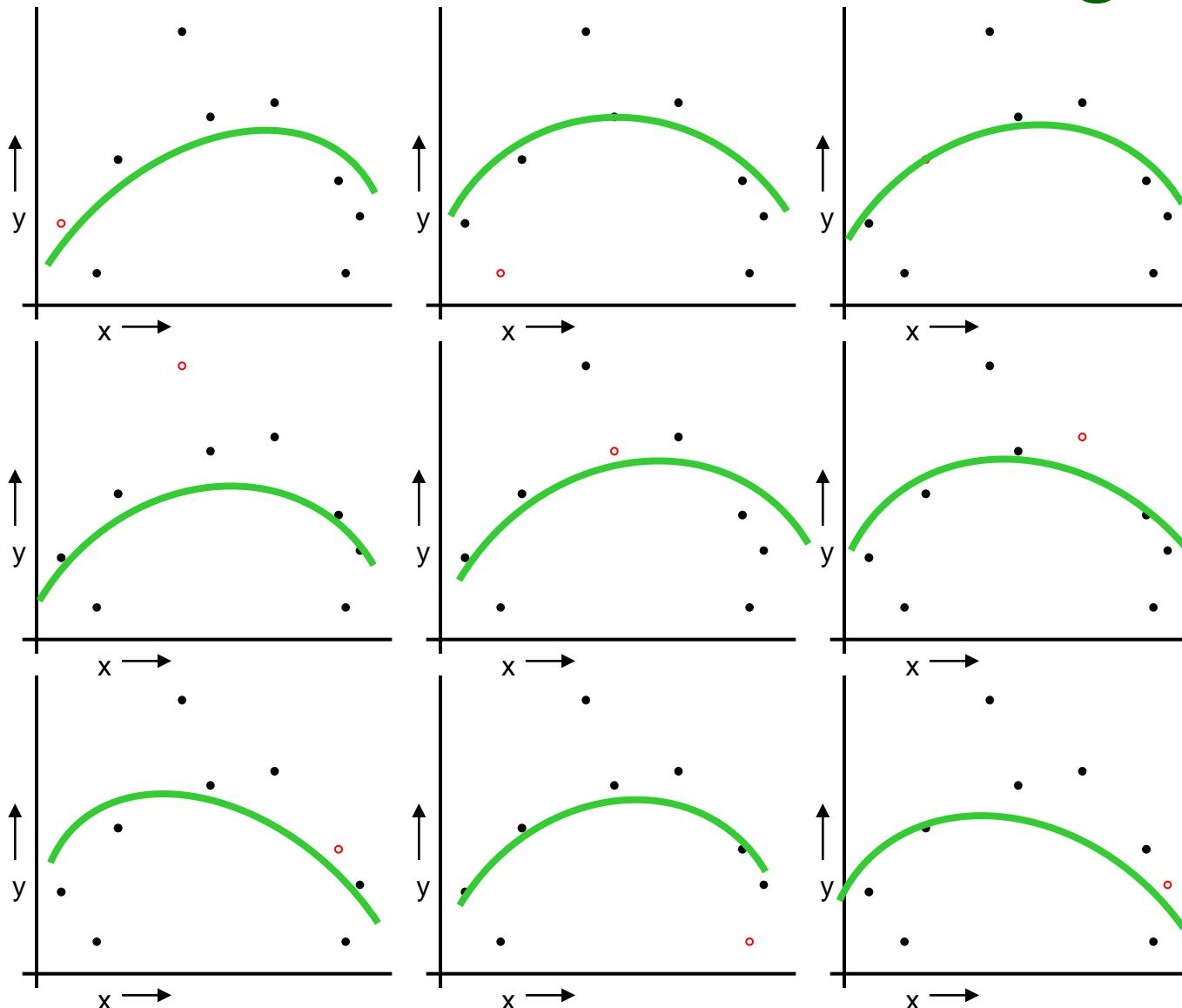
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{LOOCV} = 2.12$$

LOOCV for Quadratic Regression



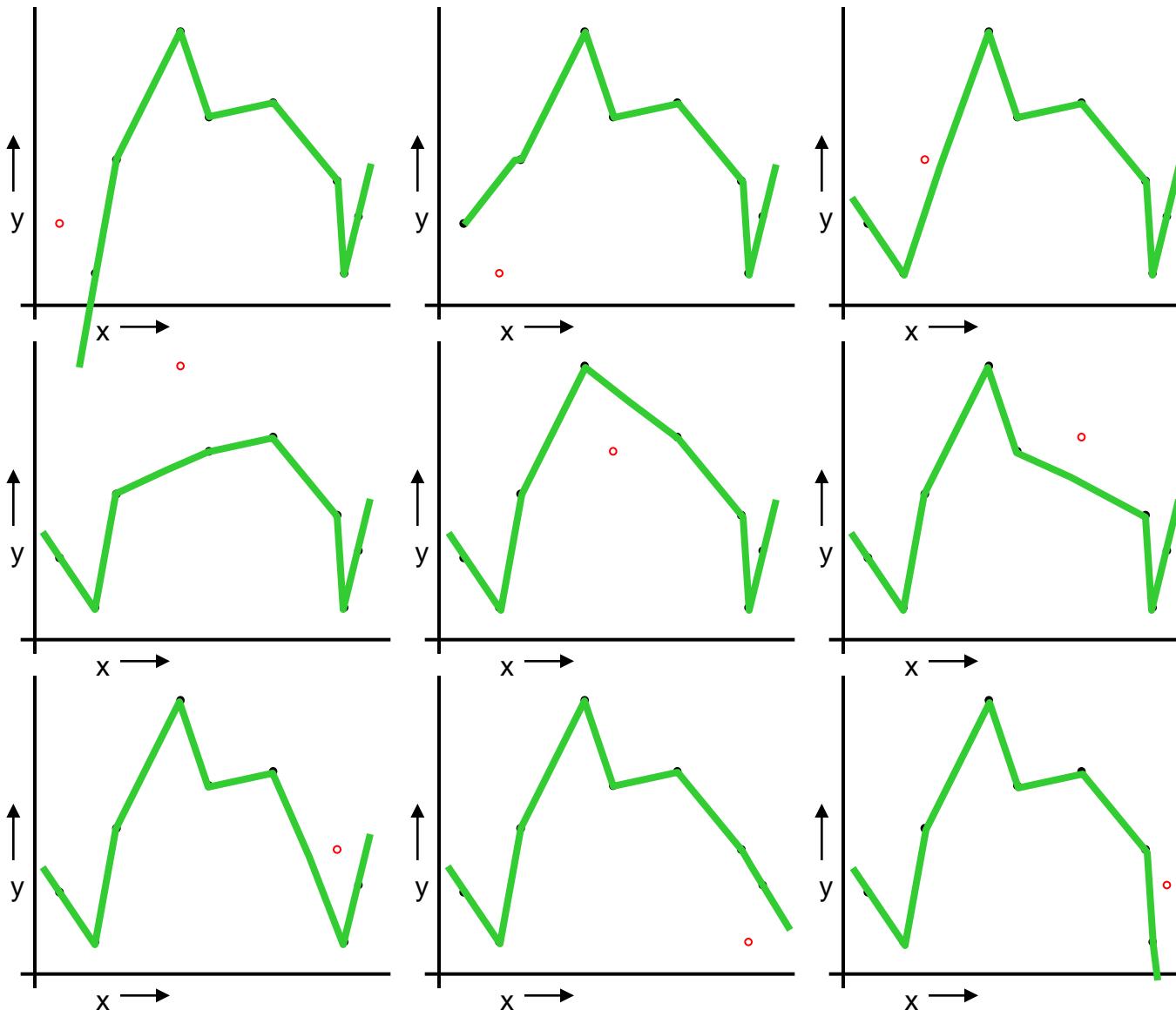
For k=1 to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{\text{LOOCV}} = 0.962$$

LOOCV for Join The Dots



For $k=1$ to R

1. Let (x_k, y_k) be the k^{th} record
2. Temporarily remove (x_k, y_k) from the dataset
3. Train on the remaining $R-1$ datapoints
4. Note your error (x_k, y_k)

When you've done all points, report the mean error.

$$MSE_{\text{LOOCV}} = 3.33$$

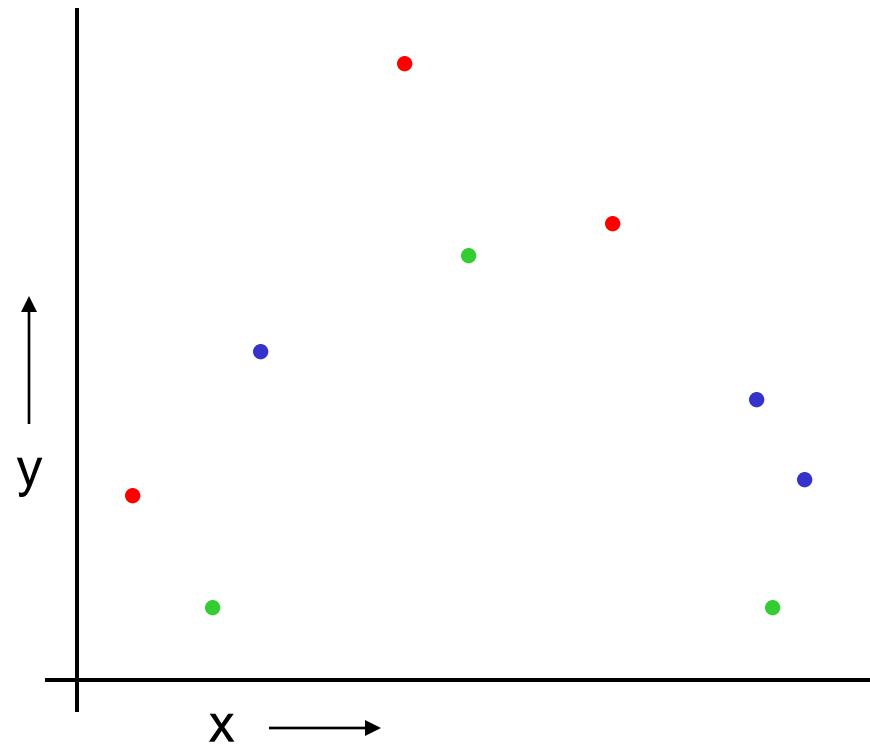
Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive. Has some weird behavior	Doesn't waste data

..can we get the best of both worlds?

k-fold Cross Validation

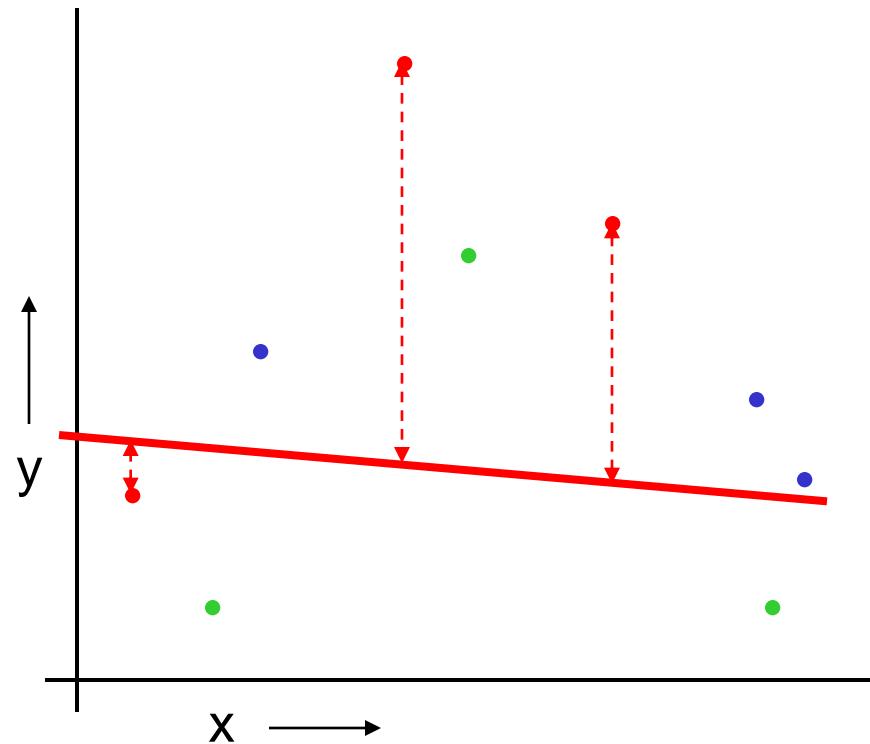
Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)



k-fold Cross Validation

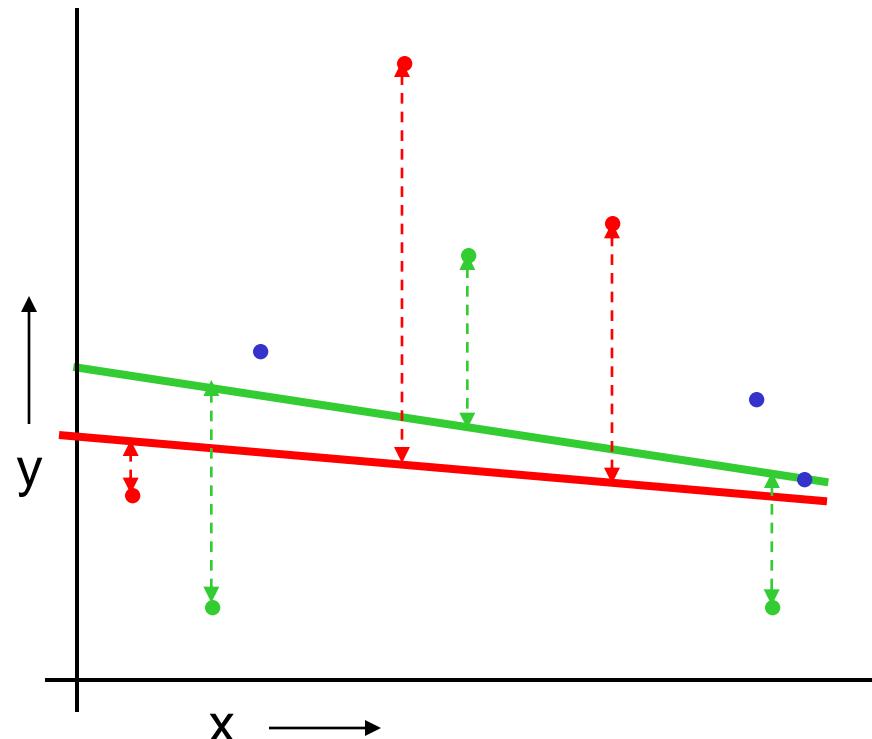
Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.



k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

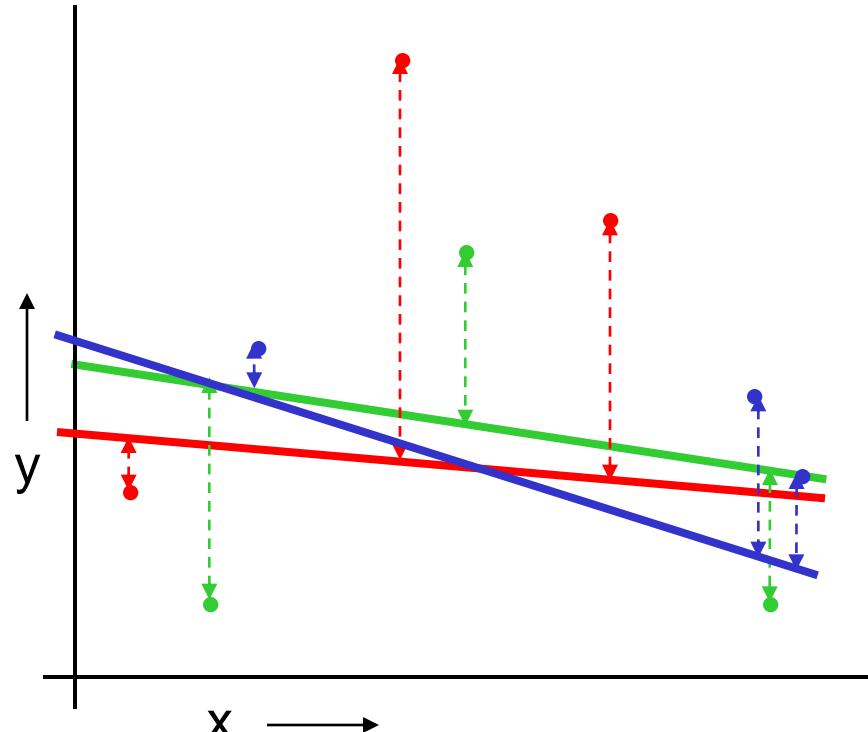


For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

k-fold Cross Validation

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

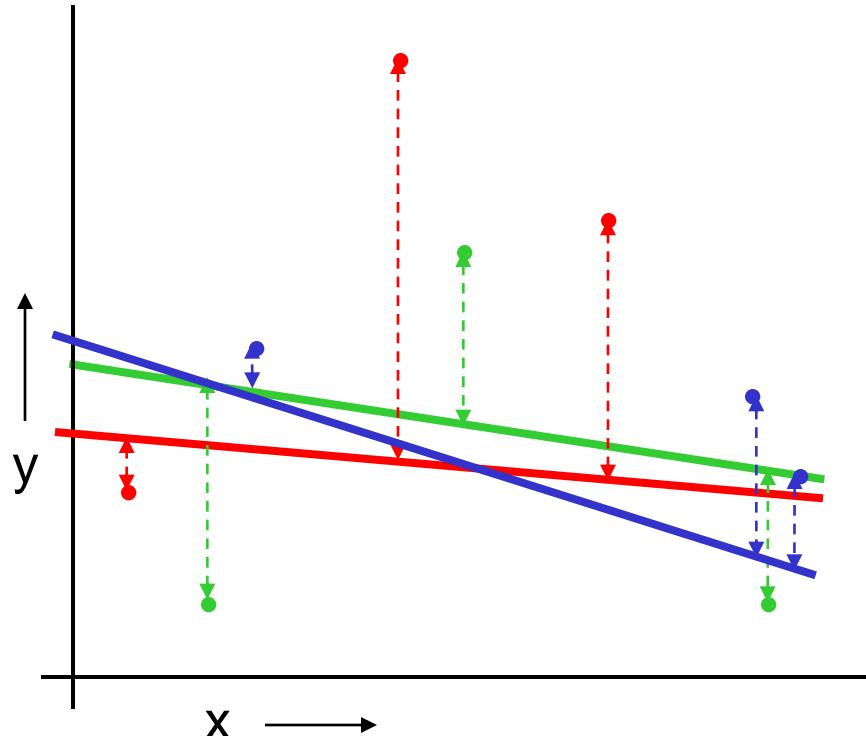


For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

k-fold Cross Validation



Linear Regression
 $MSE_{3FOLD}=2.05$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

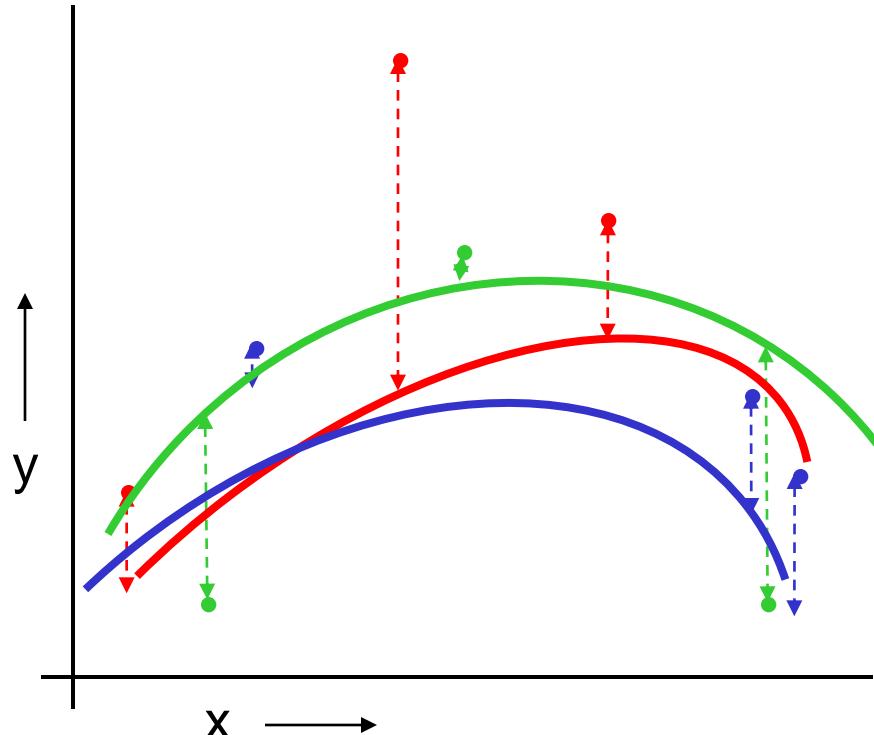
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k-fold Cross Validation



Quadratic Regression
 $MSE_{3FOLD}=1.11$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

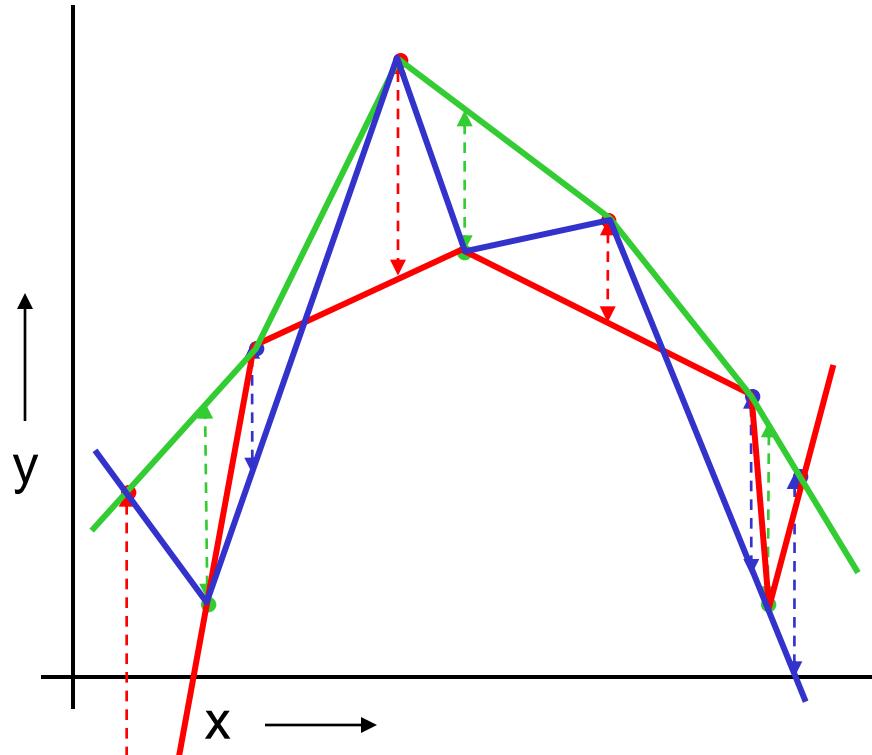
For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

k-fold Cross Validation



Joint-the-dots
 $MSE_{3FOLD}=2.93$

Randomly break the dataset into k partitions (in our example we'll have k=3 partitions colored Red Green and Blue)

For the red partition: Train on all the points not in the red partition. Find the test-set sum of errors on the red points.

For the green partition: Train on all the points not in the green partition. Find the test-set sum of errors on the green points.

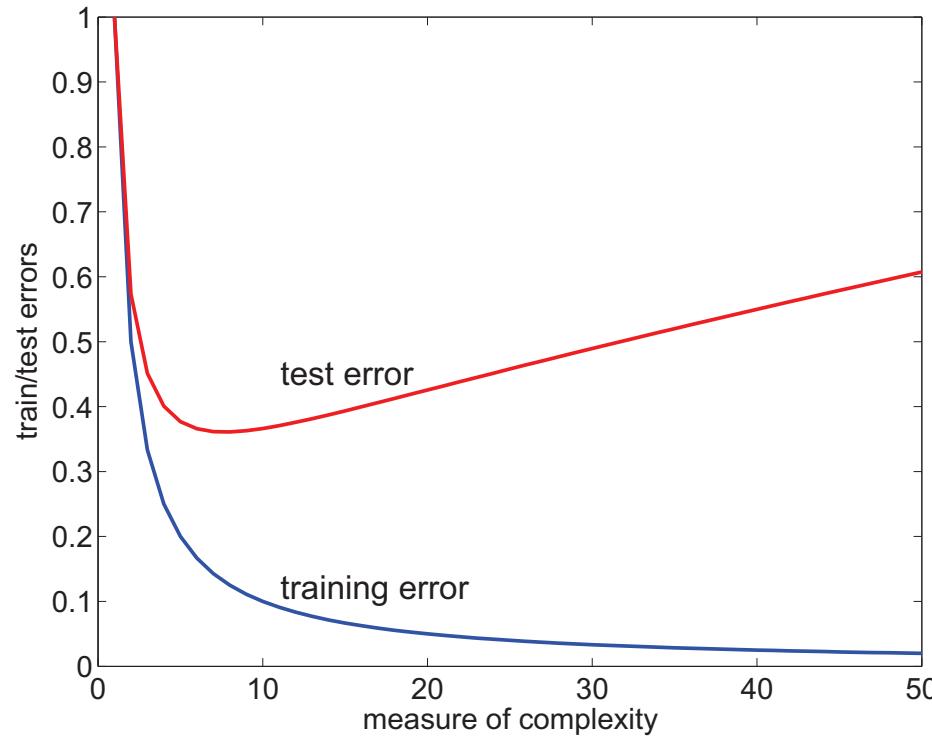
For the blue partition: Train on all the points not in the blue partition. Find the test-set sum of errors on the blue points.

Then report the mean error

Which kind of Cross Validation?

	Downside	Upside
Test-set	Variance: unreliable estimate of future performance	Cheap
Leave-one-out	Expensive. Has some weird behavior	Doesn't waste data
10-fold	Wastes 10% of the data. 10 times more expensive than test set	Only wastes 10%. Only 10 times more expensive instead of R times.
3-fold	Wastier than 10-fold. Expensivier than test set	Slightly better than test-set
R-fold	Identical to Leave-one-out	

Why care about “complexity”?



- We need a quantitative measure of complexity in order to be able to relate the training error (which we can observe) and the test error (that we'd like to optimize)



Finite case

- We'll start by considering only a finite number of possible classifiers, $h_1(\mathbf{x}), \dots, h_M(\mathbf{x})$ (e.g., randomly chosen linear classifiers)
- Key questions:
 1. Given n training examples and M possible classifiers how far can the training and test errors be?
 2. How many training examples do we need so that the errors are close?

The answers will depend on M .



Finite case: definitions

$$\hat{\mathcal{E}}_n(i) = \frac{1}{n} \sum_{t=1}^n \overbrace{\text{Loss}(y_t, h_i(\mathbf{x}_t))}^{=0, 1} = \text{empirical error of } h_i(\mathbf{x})$$

$$\mathcal{E}(i) = E_{(\mathbf{x}, y) \sim P} \{ \text{Loss}(y, h_i(\mathbf{x})) \} = \text{expected error of } h_i(\mathbf{x})$$



Finite case: definitions

$$\hat{\mathcal{E}}_n(i) = \frac{1}{n} \sum_{t=1}^n \overbrace{\text{Loss}(y_t, h_i(\mathbf{x}_t))}^{=0, 1} = \text{empirical error of } h_i(\mathbf{x})$$

$$\mathcal{E}(i) = E_{(\mathbf{x}, y) \sim P} \{ \text{Loss}(y, h_i(\mathbf{x})) \} = \text{expected error of } h_i(\mathbf{x})$$

- Suppose we choose the classifier that minimizes the training error, $\hat{i}_n = \arg \min_{i=1, \dots, M} \hat{\mathcal{E}}_n(i)$, then

$$\text{Training error} = \hat{\mathcal{E}}_n(\hat{i}_n)$$

$$\text{Test error} = \mathcal{E}(\hat{i}_n)$$



Finite case: errors

- The training and test errors,

$$\text{Training error} = \hat{\mathcal{E}}_n(\hat{i}_n)$$

$$\text{Test error} = \mathcal{E}(\hat{i}_n)$$

are necessarily close if we can show that the errors are close for all the classifiers in our set:

$$|\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| \leq \epsilon, \text{ for all } i = 1, \dots, M$$

- We can now express our key questions more formally in terms of n , M , and ϵ



Finite case: key questions revisited

- Key questions (rewritten):
 1. Given n training examples and M possible classifiers, what is the smallest ϵ such that

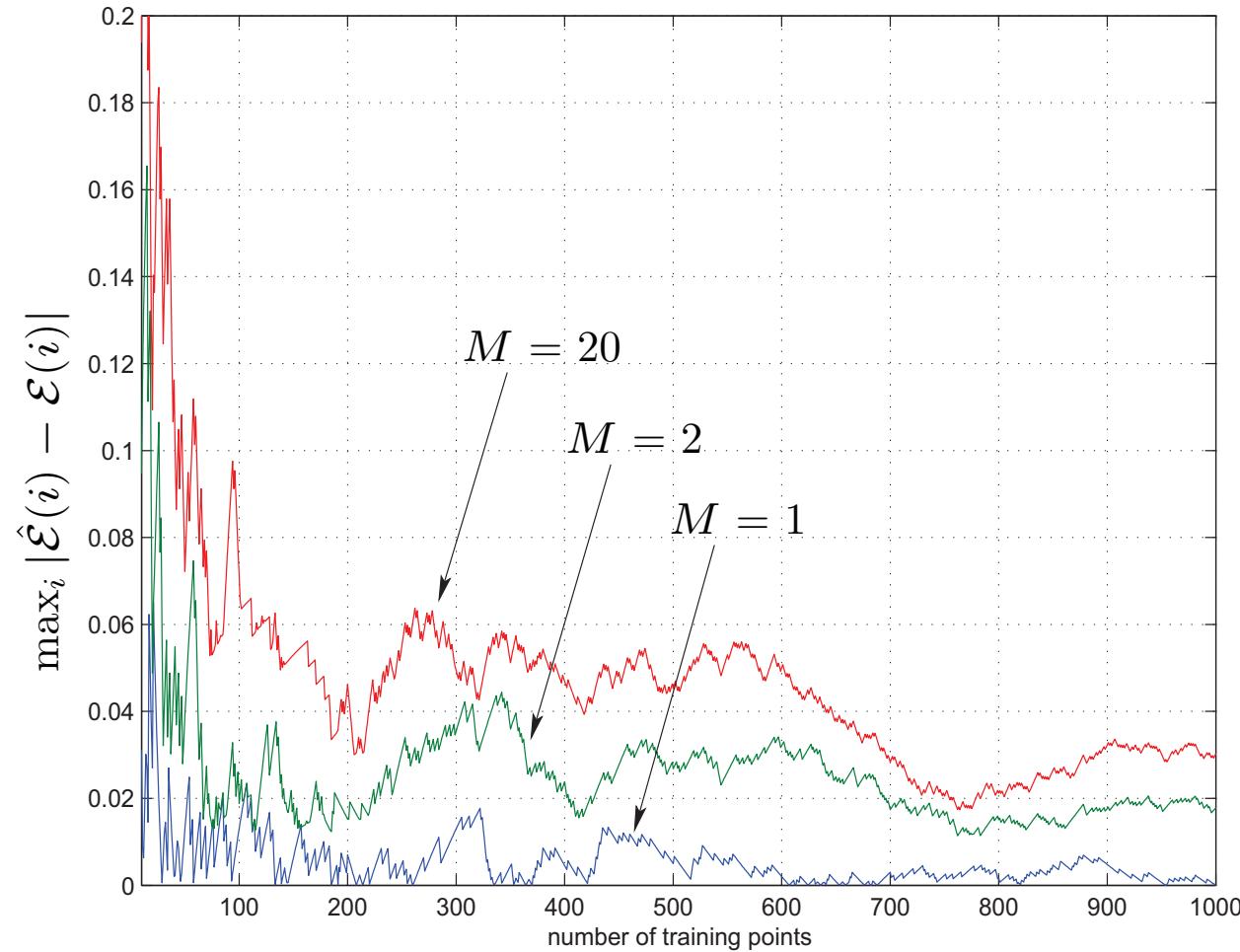
$$\max_{i=1,\dots,M} |\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| \leq \epsilon$$

- 2. For a given ϵ how many training examples do we need so that

$$\max_{i=1,\dots,M} |\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| \leq \epsilon$$

Since training examples are sampled at random from some underlying distribution, we can only answer these questions probabilistically.

Finite case: errors





Finite case: probabilistic statement

- We can relate n , M , and ϵ by requiring that with high probability, the empirical errors of all the classifiers in our set are ϵ -close to their expected errors:

$$P\left(\max_{i=1,\dots,M} |\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| \leq \epsilon\right) \geq 1 - \delta$$

The probability is taken over the choice of the training set and $1 - \delta$ specifies our confidence in the probabilistic statement.



Finite case: probabilistic statement

- We can relate n , M , and ϵ by requiring that with high probability, the empirical errors of all the classifiers in our set are ϵ -close to their expected errors:

$$P\left(\max_{i=1,\dots,M} |\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| \leq \epsilon\right) \geq 1 - \delta$$

The probability is taken over the choice of the training set and $1 - \delta$ specifies our confidence in the probabilistic statement.

- Equivalently, we can bound the probability that the empirical error of some classifier in our set deviates more than ϵ from the expected error:

$$P\left(\max_{i=1,\dots,M} |\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| > \epsilon\right) \leq \delta$$



Finite case cont'd

- Let's fix n , M , and ϵ and try to find δ so that

$$P\left(\max_{i=1,\dots,M} |\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| > \epsilon\right) \leq \delta$$

still holds. The probability is taken over the choice of the training set.

By using the fact that $P(A \text{ or } B) \leq P(A) + P(B)$ we get

$$\begin{aligned} P\left(\max_i |\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| > \epsilon\right) &\leq \sum_{i=1}^M P\left(|\hat{\mathcal{E}}_n(i) - \mathcal{E}(i)| > \epsilon\right) \\ &\leq \sum_{i=1}^M 2 \exp(-2n\epsilon^2) \quad (\text{Chernoff}) \\ &= M \cdot 2 \exp(-2n\epsilon^2) = \delta \end{aligned}$$



Finite case cont'd

- We are now able to relate n , M , ϵ , and δ :

$$M \cdot 2 \exp(-2n\epsilon^2) = \delta, \quad \text{or} \quad \epsilon = \sqrt{\frac{\log(M) + \log(2/\delta)}{2n}}$$

- We can restate our result in terms of a bound on the expected error of any classifier in our set.

Theorem: With probability at least $1 - \delta$ over the choice of the training set, for all $i = 1, \dots, M$

$$\mathcal{E}(i) \leq \hat{\mathcal{E}}_n(i) + \epsilon(n, M, \delta)$$

where $\epsilon = \epsilon(n, M, \delta)$ is a “complexity penalty”.



Measures of complexity

- Typically the set of classifiers is not a finite nor a countable set (e.g., the set of linear classifiers)
- There are still many ways of trying to capture the “effective” number of classifiers in such a set:
 - degrees of freedom (number of parameters)
 - Vapnik-Chervonenkis (VC) dimension
 - description length
 - etc.



VC-dimension: preliminaries

- **A set of classifiers F :** For example, this could be the set of all possible linear classifiers, where $h \in F$ means that

$$h(\mathbf{x}) = \text{sign} (w_0 + \mathbf{w}_1^T \mathbf{x})$$

for some values of the parameters w_0, \mathbf{w}_1 .



VC-dimension: preliminaries

- **Complexity:** how many different ways can we label n training points $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ with classifiers $h \in F$?

In other words, how many distinct binary vectors

$$[h(\mathbf{x}_1) \ h(\mathbf{x}_2) \ \dots \ h(\mathbf{x}_n)]$$

do we get by trying out each $h \in F$ in turn?

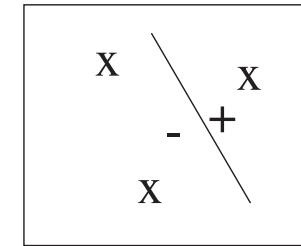
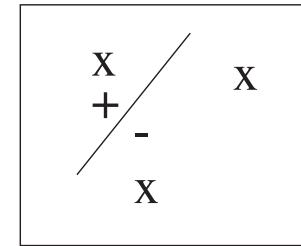
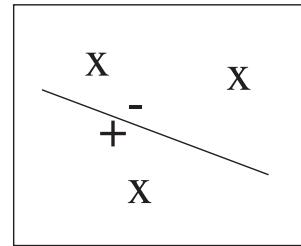
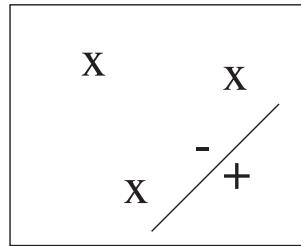
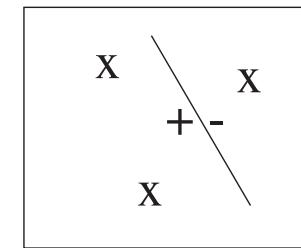
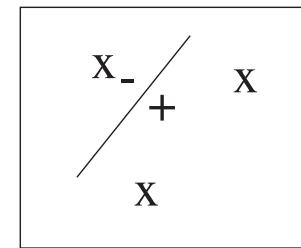
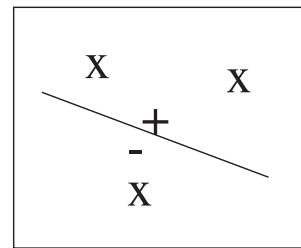
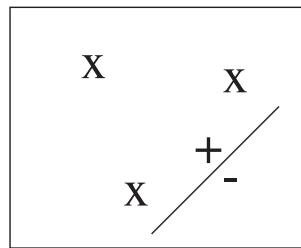
$$\begin{bmatrix} -1 & 1 & \dots & 1 \end{bmatrix} \ h_1$$
$$\begin{bmatrix} 1 & -1 & \dots & 1 \end{bmatrix} \ h_2$$

...

VC-dimension: shattering

- A set of classifiers F *shatters* n points $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ if

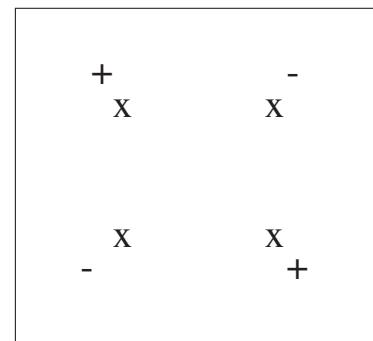
$$[h(\mathbf{x}_1) \ h(\mathbf{x}_2) \ \dots \ h(\mathbf{x}_n)], \ h \in F$$
 generates all 2^n distinct labelings.
- Example: linear decision boundaries shatter (any) 3 points in 2D



but not any 4 points...

VC-dimension: shattering cont'd

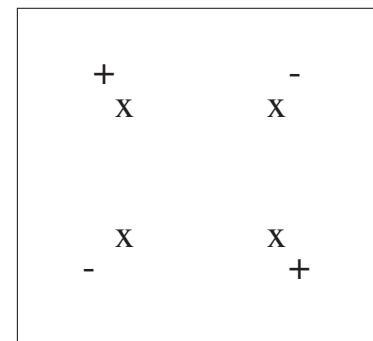
- We cannot shatter any set of 4 points in 2D with linear classifiers. For example, we cannot generate the following XOR-labeling:



- More generally: the set of all d -dimensional linear classifiers can shatter exactly $d + 1$ points

VC-dimension: shattering cont'd

- We cannot shatter any set of 4 points in 2D with linear classifiers. For example, we cannot generate the following XOR-labeling:



- More generally: the set of all d -dimensional linear classifiers can shatter exactly $d + 1$ points
- **Definition:** The VC-dimension d_{VC} of a set of classifiers F is the number of points F can shatter



Learning and VC-dimension

- We learn something only after we no longer can shatter the training points (have more than d_{VC} training examples)

Rationale: suppose we have n training examples and labels $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ and $n < d_{VC}$. Does the training set constrain our prediction for \mathbf{x}_{n+1} ?

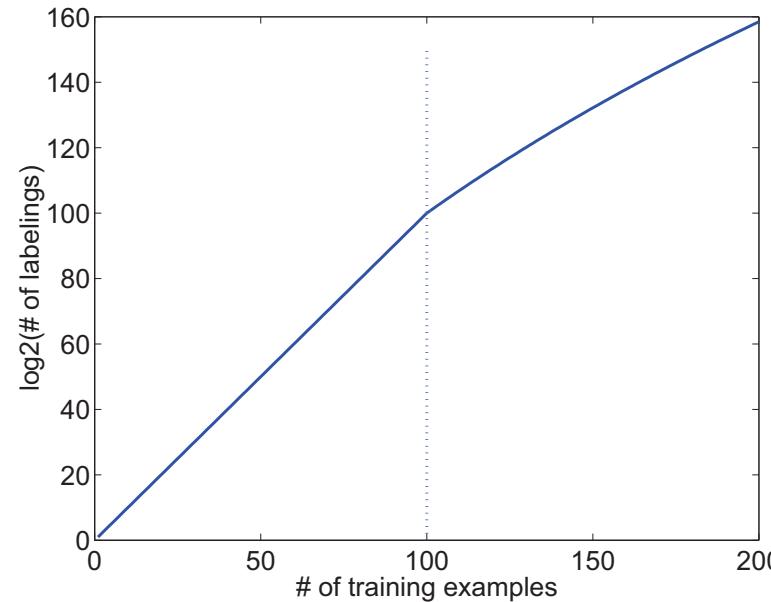
Because we expect to be able to shatter $n+1$ points ($\leq d_{VC}$) it follows that we can find $h_1, h_2 \in F$, both consistent with training labels, but

$$h_1(\mathbf{x}_{n+1}) = 1, \quad h_2(\mathbf{x}_{n+1}) = -1$$

We therefore cannot determine which label to predict for \mathbf{x}_{n+1} .

Learning and VC-dimension

- We learn something only after we no longer can shatter the training points (have more than d_{VC} training examples)



$$n \leq d_{VC} : \quad \# \text{ of labelings} = 2^n$$

$$n > d_{VC} : \quad \# \text{ of labelings} \leq \left(\frac{en}{d_{VC}} \right)^{d_{VC}}$$

Learning and VC-dimension

- By essentially replacing $\log M$ in the finite case with the log of the number of possible labelings by the set of classifiers over n (really $2n$) points, we get an analogous result:

Theorem: With probability at least $1 - \delta$ over the choice of the training set, for all $h \in F$

$$\varepsilon(h) \leq \hat{\varepsilon}_n(h) + \xi(n, d_{VC}, \delta)$$

$$\xi(n, d_{VC}, \delta) = \sqrt{\frac{d_{VC} \left(\log \frac{2n}{d_{VC}} + 1 \right) + \log \frac{4}{\delta}}{n}}$$

Unfortunately, a loose bound

Model selection

- We try to find the model with the best balance of complexity and the fit to the training data
- Ideally, we would select a model from a nested sequence of models of increasing complexity

Model 1 d_1

Model 2 d_2

Model 3 d_3

where $d_1 \leq d_2 \leq d_3 \leq \dots$

- Basic model selection criterion:

Criterion = (empirical) score + Complexity penalty

Structural risk minimization

- In structural risk minimization we define the models in terms of VC-dimension (or refinements)

Model 1 $d_{VC} = d_1$

Model 2 $d_{VC} = d_2$

Model 3 $d_{VC} = d_3$

where $d_1 \leq d_2 \leq d_3 \leq \dots$

- The selection criterion: lowest upper *bound* on the expected loss

Expected loss \leq Empirical loss + Complexity penalty

Example

- Models of increasing complexity

$$\text{Model 1 } K(\mathbf{x}_1, \mathbf{x}_2) = (1 + (\mathbf{x}_1^T \mathbf{x}_2))$$

$$\text{Model 2 } K(\mathbf{x}_1, \mathbf{x}_2) = (1 + (\mathbf{x}_1^T \mathbf{x}_2))^2$$

$$\text{Model 3 } K(\mathbf{x}_1, \mathbf{x}_2) = (1 + (\mathbf{x}_1^T \mathbf{x}_2))^3$$

⋮ ⋮

- These are nested, i.e.,

$$F_1 \subseteq F_2 \subseteq F_3 \subseteq \dots$$

where F_k refers to the set of possible decision boundaries that the model k can represent.

- Still need to derive the criterion...

Structural risk minimization cont'd

- For our zero-one loss (classification error), we can derive the following complexity penalty (Vapnik 1995):

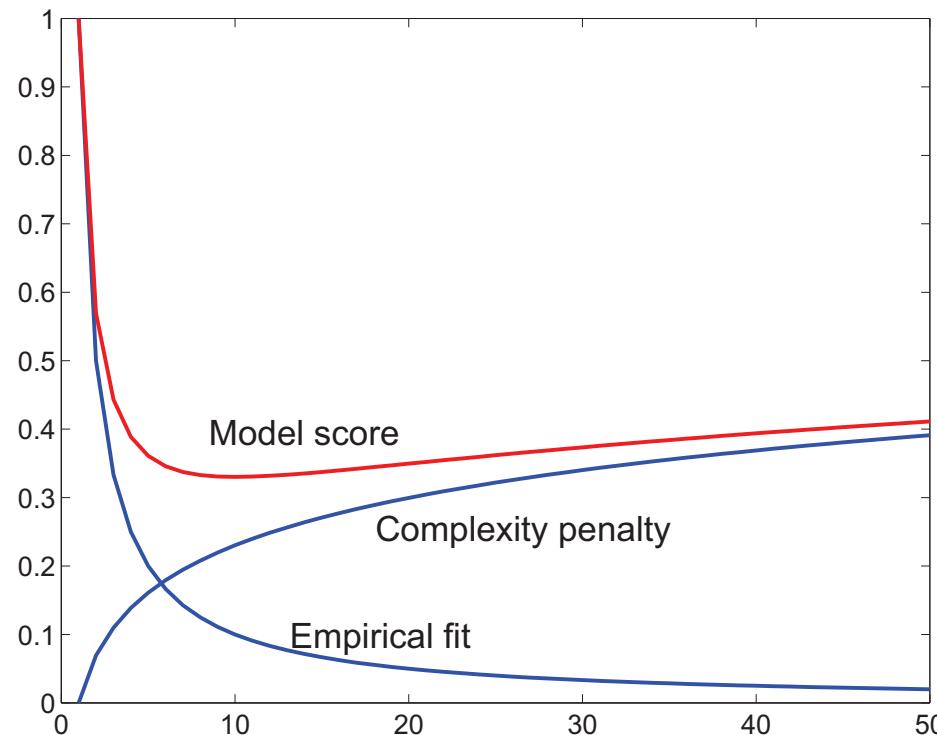
$$\epsilon(n, \delta, d) = \sqrt{\frac{d_{VC}(\log(2n/d_{VC}) + 1) + \log(1/(4\delta))}{n}}$$

1. This is an increasing function of d_{VC}
2. Increases as δ decreases
3. Decreases as a function of n

(this is not the only choice...)

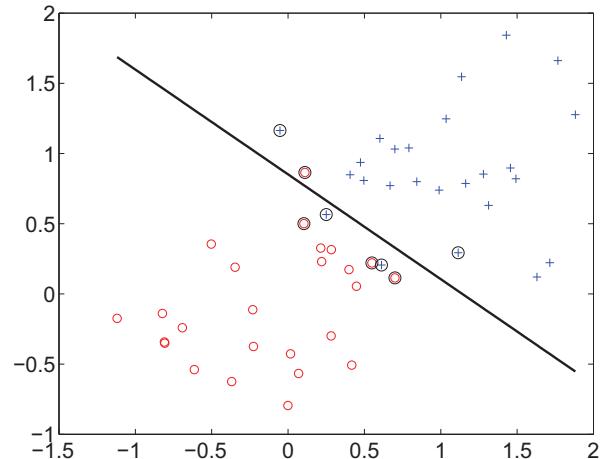
Structural risk minimization cont'd

- Competition of terms...
 1. Empirical loss decreases with increasing d_{VC}
 2. Complexity penalty increases with increasing d_{VC}

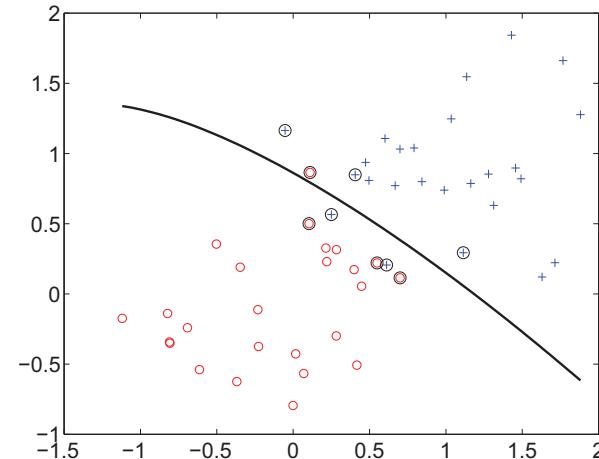


- We find the minimum of the model score (bound).

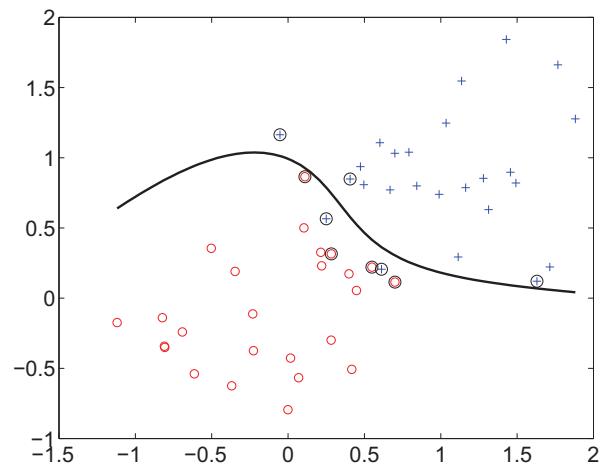
Structural risk minimization: example



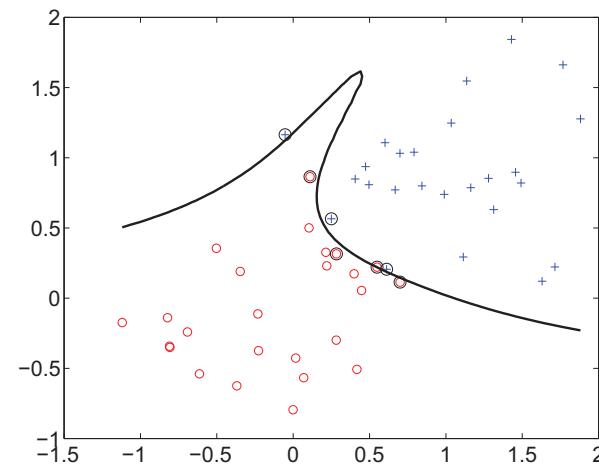
linear



2nd order polynomial



4th order polynomial



8th order polynomial

Structural risk minimization: example cont'd

- Number of training examples $n = 50$, confidence parameter $\delta = 0.05$.

Model	d_{VC}	Empirical fit	Complexity penalty $\epsilon(n, \delta, d_{VC})$
1 st order	3	0.06	0.5501
2 nd order	6	0.06	0.6999
4 th order	15	0.04	0.9494
8 th order	45	0.02	1.2849

- Structural risk minimization would select the simplest (linear) model in this case.

Example: VC dimension of 1-dimensional intervals

- $X = \mathbb{R}$ (e.g., heights of people)
- H is the set of hypotheses of the form $a < x < b$
- Subset containing two instances $S = \{3.1, 5.7\}$



- Can S be shattered by H ?
- Yes, e.g., $(1 < x < 2), (1 < x < 4), (4 < x < 7), (1 < x < 7)$
- Since we have found a set of two that can be shattered, $VC(H)$ is at least two
- However, no subset of size three can be shattered



- Therefore $VC(H) = 2$
- Here $|H|$ is infinite but $VC(H)$ is finite

2. (T/F) If there exists a set of k instances that cannot be shattered by H , then $VC(H) < k$.

3. Give the VC dimension of the class:

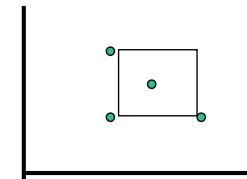
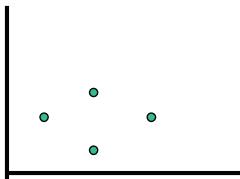
H is the set of all perceptrons in 2D plane, i.e.

$H = \{h_w | h_w = \theta(w_0 + w_1x_1 + w_2x_2) \text{ where } \theta(z) = 1 \text{ iff } z \geq 0 \text{ otherwise } \theta_z = 0\}$.

4. $H = \text{Axis parallel rectangles in } \mathbb{R}^2$

What is the VC dimension of H ?

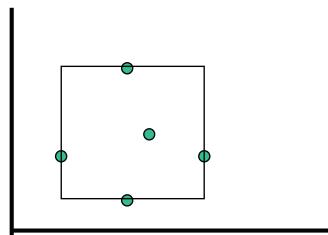
Some four instance can be shattered and some cannot



(need to consider here 16 different rectangles)

Shows that $VC(H) \geq 4$

- But, no five instances can be shattered



Since, there can be at most 4 distinct extreme points (smallest or largest along some dimension) and these cannot be included (labeled +) without including the 5th point.

Therefore $VC(H) = 4$